**DOCTOR OF SCIENCES**

# Filter-trust-region methods for nonlinear optimization

Sainvitu, Caroline

*Award date:*
2007

*Awarding institution:*
University of Namur


**FACULTES UNIVERSITAIRES NOTRE-DAME DE LA PAIX NAMUR**

FACULTE DES SCIENCES

DEPARTEMENT DE MATHEMATIQUE

# Filter-Trust-Region Methods for Nonlinear Optimization

Dissertation présentée par
**Caroline Sainvitu**
pour l'obtention du grade
de Docteur en Sciences

Composition du Jury:

Nick GOULD
Annick SARTENAER
Jean-Jacques STRODIOT
Philippe TOINT (Promoteur)
Luís VICENTE

2007

Imprimé en Belgique

ISBN-13 : 978-2-87037-548-8
Dépôt légal: D / 2007 / 1881 / 11

Facultés Universitaires Notre-Dame de la Paix
Faculté des Sciences
rue de Bruxelles, 61, B-5000 Namur, Belgium

Facultés Universitaires Notre-Dame de la Paix
Faculté des Sciences
Rue de Bruxelles, 61, B-5000 Namur, Belgium

**Méthodes de filtre et de région de confiance pour l'optimisation non-linéaire**

par Caroline Sainvitu

**Résumé:** Ce travail a pour objet l'étude théorique et l'implémentation d'algorithmes permettant la résolution de deux types particuliers de problèmes d'optimisation non-linéaire, à savoir les problèmes d'optimisation sans contrainte et avec contraintes de bornes. Pour l'optimisation sans contrainte, nous développons un nouvel algorithme qui utilise une technique de filtre et une méthode de type région de confiance dans le but de garantir une convergence globale et d'améliorer l'efficacité des approches traditionnelles. Nous analysons également l'effet de dérivées premières et secondes approximées sur la performance de l'algorithme de filtre et de région de confiance. Nous étendons ensuite notre algorithme aux problèmes d'optimisation avec contraintes de bornes en combinant ces idées avec une méthode de projection du gradient. Des résultats numériques accompagnent les méthodes proposées et indiquent qu'elles sont compétitives par rapport aux méthodes de région de confiance plus classiques.

**Filter-trust-region methods for nonlinear optimization**

by Caroline Sainvitu

**Abstract:** This work is concerned with the theoretical study and the implementation of algorithms for solving two particular types of nonlinear optimization problems, namely unconstrained and simple-bound constrained optimization problems. For unconstrained optimization, we develop a new algorithm which uses a filter technique and a trust-region method in order to enforce global convergence and to improve the efficiency of traditional approaches. We also analyze the effect of approximate first and second derivatives on the performance of the filter-trust-region algorithm. We next extend our algorithm to simple-bound constrained optimization problems by combining these ideas with a gradient-projection method. Numerical results follow the proposed methods and indicate that they are competitive with more classical trust-region algorithms.

Dissertation doctorale en Sciences mathématiques (Ph.D. thesis in Mathematics)
Date: 17-04-2007
Département de Mathématique
Promoteur (Advisor): Prof. Ph. L. TOINT

# *Remerciements*

Je tiens tout d'abord à exprimer toute ma reconnaissance à Philippe Toint, mon promoteur, pour m'avoir acceuillie au sein de l'Unité d'Analyse Numérique ainsi que pour son encadrement, ses nombreux conseils et son soutien constant tout au long de cette thèse. Je le remercie également de m'avoir donné l'opportunité de participer à plusieurs conférences internationales. Ce fut pour moi une chance d'y présenter à chaque fois notre travail et d'y rencontrer de nombreux chercheurs.

Je tiens également à remercier Nick Gould pour sa collaboration à une partie de ce travail et pour avoir accepté d'être dans le jury de cette thèse.
Merci aussi à Annick Sartenaer, Jean-Jacques Strodiot et Luís Vicente d'avoir accepté de faire partie du jury de cette thèse.

Je tiens aussi à remercier Dominique Orban de m'avoir invitée à plusieurs reprises à présenter mes recherches lors de conférences ainsi que tous les chercheurs que j'ai pu y rencontrer pour les discussions intéressantes ainsi que les moments de détente. Je remercie notamment Andreas Wächter pour ses suggestions pertinentes.

Merci tout particulier à Katia Demaseure pour son amitié, sa bonne humeur et sa disponibilité. Je la remercie vivement pour toutes ces années passées ensemble au bout du couloir.
Merci aussi à mes anciens collègues d'Analyse Numérique, à savoir Benoît Colson, qui fut toujours présent dans les moments de doutes, et Fabian Bastin, dont la distraction légendaire nous a valu beaucoup de fous rires.
Mes plus chaleureux remerciements s'adressent également à tous *les copains du midi* avec qui j'ai eu la chance de partager pas mal de repas, pauses café et sorties. Merci à Charlotte Beauthier, Florent Deleflie, Anne-Sophie Libert, Dimitri Tomanos, Stéphane Valk, Emilie Wanufelle, Melissa Weber Mendonça, Sebastian Xhonneux et tous les autres.
Mes remerciements vont aussi vers tous les membres du département de Mathématique pour leur accueil et leur convivialité durant ces cinq années. Merci notamment à Eric Cornélis,

Murielle Haguinet, Pascale Hermans et Martine Van Caenegem.

Mes remerciements s'adressent également à Jean-Claude Lion, mon professeur de Mathématique de secondaire, qui est sans nul doute à l'origine de mon goût pour les mathématiques.

Merci aussi à tous mes amis, mathématiciens ou non, qui m'ont aidée, parfois à leur insu, par tous ces moments de détente passés en leur compagnie.
Je tiens également à remercier mes parents qui ont toujours été présents dans les moments difficiles et sans qui je ne serais pas où j'en suis aujourd'hui.

Enfin, pour tout le reste et bien plus encore, je remercie Olivier Jacquet. Merci d'avoir été à mes côtés durant la longue et difficile période de la rédaction de cette thèse. Merci de m'avoir écoutée patiemment, d'avoir supporté les sautes d'humeur très fréquentes ces derniers mois et les remises en cause.

A tous, encore merci.

Caroline

# Contents

**Introduction** **vii**

**1 Generalities on optimization** **1**
1.1 What is optimization? . . . . . 1
1.2 Mathematical programming . . . . . 2
1.2.1 Formulation . . . . . 2
1.2.2 Classification of mathematical programs . . . . . 3
1.3 Basic notions . . . . . 6
1.3.1 Mathematical background . . . . . 6
1.3.2 Generalities on convergence . . . . . 9
1.3.3 Derivatives . . . . . 11
1.3.4 Approximation to derivatives . . . . . 13
1.3.5 Automatic differentiation . . . . . 15

**2 Background on nonlinear optimization** **17**
2.1 Characterization of solutions . . . . . 17
2.2 Optimality conditions . . . . . 18
2.2.1 Unconstrained optimization . . . . . 19
2.2.2 Constrained optimization . . . . . 20
2.2.3 Criticality measure . . . . . 24
2.3 Methods for nonlinear unconstrained optimization . . . . . 24
2.3.1 Local methods . . . . . 24
2.3.2 Line-search methods . . . . . 29
2.3.3 Trust-region methods . . . . . 29
2.3.4 Conjugate-gradient methods . . . . . 32
2.4 Methods for nonlinear constrained optimization . . . . . 33
2.4.1 Penalty methods . . . . . 34
2.4.2 Augmented Lagrangian methods . . . . . 36

2.4.3 Sequential Quadratic Programming . . . . . 38
2.5 References . . . . . 40

**3 A quick survey of filter methods 41**
3.1 Motivation of the filter . . . . . 41
3.2 The first filter approach . . . . . 42
3.3 Bibliographical review . . . . . 47

**I Unconstrained Optimization 49**

**4 A filter-trust-region method for unconstrained optimization 51**
4.1 The problem and the new algorithm . . . . . 52
4.1.1 Computing a trial point . . . . . 53
4.1.2 The multidimensional filter . . . . . 54
4.1.3 The filter-trust-region algorithm . . . . . 56
4.2 Convergence analysis . . . . . 58
4.2.1 Assumptions and notations . . . . . 58
4.2.2 Convergence to first-order critical points . . . . . 60
4.2.3 Analysis of a counter-example . . . . . 66
4.2.4 Convergence to second-order critical points . . . . . 70
4.3 Conclusion . . . . . 72

**5 Numerical results 73**
5.1 Testing environment . . . . . 73
5.2 Performance profiles . . . . . 76
5.3 Practical aspects . . . . . 77
5.4 Performance and comparisons . . . . . 80
5.4.1 Filter versus pure trust-region variants . . . . . 80
5.4.2 Comparison on quadratic programs . . . . . 84
5.4.3 Comparison with LANCELOT-B . . . . . 87
5.4.4 Algorithmic variants . . . . . 93
5.4.5 Unrestricted steps . . . . . 97
5.5 Conclusion . . . . . 99

**6 Do approximate derivatives hurt filter methods? 101**
6.1 The framework . . . . . 101
6.2 Numerical investigation . . . . . 103

6.2.1 Finite differences . . . 105
6.2.2 Secant updates . . . 113
6.2.3 Perturbation of the Hessian . . . 121
6.2.4 Comparison . . . 121
6.3 Conclusion . . . 125

**II Bound-Constrained Optimization 127**

**7 A filter-trust-region method for bound-constrained optimization 129**
7.1 Introduction to bound-constrained optimization . . . 129
7.1.1 Optimality conditions . . . 130
7.1.2 Gradient-projection method . . . 132
7.2 The new algorithm . . . 134
7.2.1 Computing a trial point . . . 135
7.2.2 The multidimensional filter . . . 138
7.2.3 The filter-trust-region algorithm . . . 139
7.3 Global convergence to first-order critical points . . . 140
7.4 Conclusion . . . 145

**8 Numerical results 147**
8.1 Testing environment . . . 147
8.2 Practical aspects . . . 148
8.3 Performance and comparisons . . . 151
8.3.1 Filter versus pure trust-region variants . . . 152
8.3.2 Comparison with LANCELOT-B . . . 157
8.3.3 Signed filter entries . . . 162
8.4 Conclusion . . . 162

**Conclusions and further research perspectives 165**

**Summary of contributions 169**

**Bibliography 171**

**Main notations and abbreviations 183**

**Index 187**

**Appendices** **191**

**A Results of** FILTRUNC **191**

**B Results of** FILTBOUND **199**

# Introduction

Optimization, also called mathematical programming, is the branch of computational science devoted to the study of problems whose aim is to determine the best allocation of possibly limited resources required to meet a given objective. So, an optimization problem involves minimizing or maximizing a function, called the *objective function*, of several variables, potentially subject to some restrictions to the values of these variables defined by a set of *constraints*. Optimization, which is a major component part of human life as well as natural processes, has been used for centuries.
Mathematicians wish to develop appropriate tools for the study of optimization problems and in particular for the characterization of their solutions, like the *optimality conditions*. Often, one cannot solve the equations of the necessary optimality conditions analytically in an efficient way and the goal of the researchers is then to design *iterative* methods for approximately solving these optimality systems. Then one looks for an algorithm that leads efficiently to a good approximation of the problem solution. When developing algorithms, the assessment of the computational performance is crucial; one wants the algorithms to be both *efficient* and *robust* but they should also have good performance in terms of accuracy, run time or computer storage. However, an important thing to realize when conceiving an algorithm for solving nonlinear programs, which are problems where the objective function and/or some of the constraints may be nonlinear, is that it is rather impossible to have one that can solve efficiently *all* problems. Besides, the ability to ensure convergence to the *global* solution of nonlinear nonconvex problems is a very difficult task on most problems.

This thesis is concerned with the design, implementation and assessment of novel algorithms for solving two particular classes of nonlinear optimization problems. More specifically, it addresses the development of efficient and robust algorithms for finding *local* solutions to *unconstrained* and *bound-constrained nonlinear* and possibly *nonconvex* mathematical programs, in the context of *trust-region* schemes and *filter* techniques, both of them having proven to be very efficient in nonlinear programming. The theoretical convergence properties of proposed algorithms are to be explored and their numerical performances are to be validated on a large

set of test problems.

## Contributions and structure of the thesis

We now give a brief description of the contents of this thesis and the contributions included therein. Chapter 1 is an introduction to optimization, it also exposes the terminology used throughout this work and the mathematical background useful for our needs. The next chapter presents some basic concepts and results from mathematical programming as well as an overview of different methods and algorithms designed for solving optimization problems. However, this survey is far from being exhaustive and essentially covers tools that will be used and discussed in the following chapters of this thesis. Chapter 3 introduces the notion of a *filter*, which is at the core of algorithms we have developed in this dissertation. We motivate the use of a filter technique, give the main ideas of algorithms based on this concept and review the existing research related to this quite recent topic.

In Part I of this manuscript, made of three chapters, we consider *unconstrained* optimization problems. This class of problems is especially interesting because a lot of methods for constrained optimization problems are designed in a way that benefit from unconstrained optimization techniques.
The classical safeguards around Newton method, as line searches or trust regions, often restrain the algorithmic efficiency. So researchers have been interested in the development of less obstructive safeguards while continuing to ensure global convergence. This leads to the relatively recent idea of the *filter* which is an interesting alternative to penalty functions used in constrained optimization. The generality of this concept allows its use, for example, in trust-region and line-search frameworks, as well as in active-set and interior-point methods. Various algorithms using filter methods have been proposed and major contributions have been made in this field in the last few years. Nowadays, some of the most effective nonlinear optimization softwares are based on filter techniques. However, most filter-based algorithms proposed in the literature are designed for constrained optimization. To our knowledge, little attention has been paid to general unconstrained optimization. We propose to use filter techniques to design an algorithm for solving unconstrained optimization problems. This results in an algorithm, called FILTRUNC, combining the efficiency of *filter* techniques and the robustness of *trust-region* methods. Our contributions are both theoretical and practical. Chapter 4 describes the new filter-trust-region algorithm; global convergence properties of our method are examined in Section 4.2. The practical performance of FILTRUNC is examined in Chapter 5. We compare our algorithm with a more classical trust-region method on a large set of test problems in terms of efficiency and reliability. Moreover, FILTRUNC is also compared with the

software LANCELOT-B. In Chapter 6, we investigate the influence of using approximate first and/or second-order derivatives of the objective function on our filter-trust-region algorithm. Finite-difference approximations and secant updates are considered and compared.

The theoretical and practical extension of the filter-trust-region algorithm for solving simple-bound-constrained optimization problems is the subject of Part II. Many real-world problems include simple bounds on their variables. Therefore, treating problems of this nature is also of importance. Moreover, unconstrained problems should often require bounds on their variables to avoid evaluations of an objective function not defined outside the box specified by these bounds.
Our contribution is the design of an algorithm which combines in a trust-region framework filter techniques and gradient-projection methods. Chapter 7 focuses on the description and the convergence theory of the method while the numerical experiments of our solver, called FILTBOUND, are reported and discussed in Chapter 8.

Finally, we conclude the thesis with a summary of the work, its contributions and some directions for future research.

# Chapter 1

# Generalities on optimization

## 1.1 What is optimization?

Optimization is a mathematical discipline which can be described as follows : *the action of finding the best solution*. Optimization problems arise in lots of aspects of human life and activities. Indeed, optimization occurs every day and everywhere when people, industries or systems want to minimize or maximize something. For instance, in a manufacturing process, one might want to maximize the profit or minimize the cost. In designing an automobile panel, one might want to maximize the strength. Airline companies might want to schedule crews and airplanes in order to minimize cost. Optimization also occurs in nature, many laws of physics being formulated as principles of minimum or maximum of some characteristics of observed objects or systems, like, for example, energy. The most important areas of application include biology, chemistry, economics, engineering, planning, physics, social management and statistics.

Optimization problems are composed of three major ingredients. An *objective function* we want to minimize or maximize and which measures the quality of solutions. A set of *unknowns* or *variables* which affect the value of the objective function. Variables are things we can change, the things we need to decide upon. The purpose is to give some values to these variables such that minimizing or maximizing the objective function. Frequently, there is a set of *constraints* that allows the variables to take some values but excludes some others. In other words, these constraints define restrictions on the variables or interrelations of many kinds. The *solution* of an optimization problem is a set of allowed values for the variables at which the objective function takes on an *optimal* value.

Most problems arising from the real world are complex and require making decisions; so optimization is needed. Determining the appropriate variables, the objective and the potential constraints so that the formalism simulates the real-life problem adequately is called the

*modelling* phase. In an optimization process, we typically start with a real problem, full of complexities and details. From this, we extract the relevant elements to create a mathematical *model* which is built with the help of a series of mathematical relationships. Then the best solution to this model can be found using a suitable optimization algorithm. If the model is accurate enough, the solution is validated and interpreted back into the real world as a solution of the real problem. It is important to note the fact that a mathematical model is never an exact representation of the reality and it does not make sense to consider it taken out of its context. It is crucial to realize that optimization models can help in finding good solutions but are not the complete solution.

Modern optimization originated in the World War II period when Georges Dantzig had to deal with the logistical issues raised by large armies having millions of men, weapons and machines. He developed an algorithm to solve linear programming (see Section 1.2.2). Originally, the term *programming*, frequently used in optimization, means *preparing a schedule of activities*.

It is important to underline the close relationship between operations research and optimization. *Operations research* may be defined as the use of mathematical models, statistics and algorithms to help in decision-making. Most of the time, operations research is applied to study complex real-world situations, typically with the aim of improving or optimizing performance. Operations research started just before the World War II. The scientists were charged to find ways to make better decisions in various fields such as schedules for training, deployment of combat units or logistical supply. After the war it began to be used for related industrial problems. Currently, some typical applications in which operations research is used are : resource and staff allocations, road traffic management, supply chain planning, etc. In almost all operations research problems appear the minimization or the maximization of a function expressing either the operating costs or the revenues subject to some constraints. Therefore, problems arising in operations research may be expressed as optimization problems and then solved by appropriate optimization methods. On the other hand, problems arising in operations research provide a collection of test problems for the optimization field.

## 1.2 Mathematical programming

### 1.2.1 Formulation

A *mathematical program* is the formulation of an optimization problem in terms of an objective function and some possible constraints. The study and the use of mathematical programs

are called *mathematical programming*.

*Unconstrained optimization problems* can be expressed in mathematical terms as

$$\min_{x \in \mathbb{R}^n} \ f(x), \tag{1.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ is the objective function. In *constrained optimization problems*, the variables $x$ are required to belong to the *feasible region* $\Omega$, which is the set of points $x$ that satisfy the constraints; that is

$$\Omega = \{x \in \mathbb{R}^n \mid x \in X, \ c_i(x) = 0, i \in \mathcal{E}, \ c_i(x) \leq 0, i \in \mathcal{I}\},$$

where $\mathcal{E}$ and $\mathcal{I}$ are two disjoint sets of indices and $c_i : \mathbb{R}^n \to \mathbb{R}$ $(i \in \mathcal{E} \cap \mathcal{I})$. The relations $c_i(x) = 0$ $(i \in \mathcal{E})$ and $c_i(x) \leq 0$ $(i \in \mathcal{I})$ are called *equality constraints* and *inequality constraints*, respectively. We consider that $X$ is a subset of $\mathbb{R}^n$ and is in the domain of the functions $f$ and $c_i$ $(i \in \mathcal{E} \cap \mathcal{I})$. This leads to the following statement of *constrained optimization problems*

$$\begin{aligned}
&\text{minimize} && f(x) && \text{(1.2a)}\\
&\text{subject to} && x \in X, && \text{(1.2b)}\\
& && c_i(x) = 0, \quad i \in \mathcal{E}, && \text{(1.2c)}\\
& && c_i(x) \leq 0, \quad i \in \mathcal{I}. && \text{(1.2d)}
\end{aligned}$$

If any point $x$ satisfies the constraints (1.2b)-(1.2d), it is said to be a *feasible point*. If the problem has no feasible points, it is called an *infeasible problem*.
More general constraints $c_i(x) \leq b$ can be rewritten as $c_i(x) - b \leq 0$. Mathematical programs also exist which are maximization problems but these can be easily reformulated as minimization problems through the transformation

$$\max_x \ f(x) = -\min_x \ -f(x).$$

### 1.2.2 Classification of mathematical programs

The optimization problem as stated in (1.2a)-(1.2d) is a very general formulation and there is a wide variety of problem classes having different structures. Mathematical programs can be classified according to the mathematical characteristics of the objective function and the constraints, if any. This classification is used to enhance the efficiency of solution methods. In the remaining part of this section, we give a non-exhaustive list of principal categories of mathematical programs.

An important distinction, that we have already pointed out, is between mathematical programs that have constraints on their variables and those which do not. A specialized case of constrained optimization that we shall consider in Chapter 7 is the *bound-constrained optimization*. In this class of mathematical programs, the only constraints have the following form

$$l_i \leq x_i \leq u_i, \quad i = 1, \ldots, n,$$

where $l_i$ and $u_i$ are *lower* and *upper bounds* on the $i$th component of $x$. Note that any of the bounds may be infinite. Without loss of generality, we assume throughout this work that $l_i < u_i$ for all $i = 1, \ldots, n$. The feasible region of such a problem is sometimes called a *box* because of its rectangular shape.

Possibly the most important feature of a mathematical program is whether it is *continuous* or *discrete*. Continuous problems are those where the constraints set is infinite and where the variables have a "continuous" character; while discrete ones are basically those that are not continuous. *Discrete optimization* problems arise when the variables occurring in the problem can only take a finite number of discrete values. This kind of problems can arise for example in scheduling. The most important class of discrete optimization is *integer programming*, where the variables must have integer values, or even only binary values. For an overview of integer programming, we refer the reader to the books of Nemhauser and Wolsey [99] and Schrijver [115].

*Linear programming* is the problem of minimizing or maximizing a linear objective function subject to linear constraints. The feasible region for a linear program is a *polytope*, that is, a convex, connected set with flat, polygonal faces. Given a polytope and a real-valued function defined on this polytope, the aim is to find a point in the polytope where the objective function has the smallest (or the largest) value. A *linear program* is usually expressed in the following matrix form which is called the *standard form*

$$\begin{array}{ll} \min & c^T x \\ \text{s.t.} & Ax = b, \\ & x \geq 0, \end{array} \tag{1.3}$$

where $x \in \mathbb{R}^n$ is the vector of unknowns, $c \in \mathbb{R}^n$ and $b \in \mathbb{R}^m$ are vectors of known coefficients and $A$ is an $m$-by-$n$ real matrix of known coefficients. Many practical problems in operations research can be expressed as linear programming problems. Some special cases of linear programming, such as network flow problems, are considered important enough to have generated much research on specialized algorithms for their solutions. Two families of solution techniques are widely used today. Linear programs can be solved using the *simplex method*, introduced by Dantzig [37], which runs along polytope edges to find the best solution. There

are many works related to the study of the simplex method; among others, we cite the books by Chvátal [23], Murtagh [97] and Nash and Sofer [98]. By contrast, *interior-point methods* visit points within the relative interior of the feasible region. Though these techniques arise from methods for nonlinear programming, developed in the nineteen sixties by Fiacco and McCormick (see [45], [46]), their application to linear programming is due to Karmarkar [84] in 1984. The Wright's monograph [129] gives a complete introduction to theoretical and practical aspects of interior-point methods for linear programming.

When the objective function $f$ is *quadratic*, that is a function of the following form

$$f(x) = g^T x + \tfrac{1}{2} x^T H x,$$

where $g \in \mathbb{R}^n$ and $H$ is a symmetric $n$-by-$n$ matrix, and the constraints $c_i$ are all linear, the problem belongs to the class of *quadratic programming*.

In *convex programming*, a problem has a convex objective function, the equality constraint functions $c_i(\cdot)$ $(i \in \mathcal{E})$ are affine and the inequality constraint functions $c_i(\cdot)$ $(i \in \mathcal{I})$ are convex. It follows that the feasible region of a convex program is a convex set (see Section 1.3.1).

If the objective function $f$ and/or some of the equality or inequality constraints are nonlinear, the mathematical program is called a *nonlinear programming* problem . So nonlinear programming is identical to linear programming, except that the relationships can have a nonlinear form, which makes the problem in general harder than a linear one. Nevertheless, nonlinearity is nearly unavoidable in many real-world problems. Nonlinear programming lies within the continuous problem class. Some reference works on this topic are those of Bertsekas [8], Conn, Gould and Toint [34], Fletcher [48], Gill, Murray, and Wright [62], Luenberger [91], Nash and Sofer [98] and Nocedal and Wright [101]. This dissertation addresses the solution of this kind of problems.

Another way in which optimization problems can be classified is through the computable information that may be available during the solution process. For example, it may occur that analytic first and second derivatives of the objective function are not available, in that case one refers to the field of *nondifferentiable optimization* or *nonsmooth optimization* (see, for example, Dem'yanov and Vasil'ev [39]).

We finally mention *stochastic programming* which is a framework for modelling optimization problems that involve uncertainty introduced by means of random variables. While *deterministic optimization* deals with known parameters, lots of real-life problems include some unknown parameters. For a more complete treatment of stochastic programming, we refer the reader to the books of Birge and Louveaux [10] and Kall and Wallace [83].

To conclude this section, we want to mention that, nowadays, there exist optimization problem analysers, like, for example, DrAmpl created by Fourer and Orban [58]. The purposes of

this meta solver are, especially, to analyse a given model, to provide bounds on expressions appearing in the model, to assess convexity of expressions arising in the model, to classify this model and to assist in the choice of an appropriate solver for the problem under study.

## 1.3 Basic notions

The remaining of this chapter will be concerned with selected basic aspects of linear algebra and real analysis that are directly relevant to studying the mathematical programs we consider in this dissertation.

### 1.3.1 Mathematical background

#### Vectors and matrices

We denote by $\mathbb{R}^n$ the real $n$-dimensional Euclidean space. Throughout this work, a *vector* is an $n$-dimensional column of the following form

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix},$$

where each $x_i$ $(1 \leq i \leq n)$ is called the $i$th component of the vector $x$. The notation $x^T$ stands for the *transpose* of $x$. For describing iterative algorithms, when there is no risk of confusion with the $k$th component of the vector $x$, we will use the notation $x_k$ to designate a vector $x$ at the $k$th iteration ($k \in \mathbb{N}$), while the $i$th component of the vector $x$ at the $k$th iteration will be denoted by $x_{k,i}$. The *inner product* of two vectors $x$ and $y \in \mathbb{R}^n$ is defined by

$$x^T y \stackrel{\text{def}}{=} \sum_{i=1}^{n} x_i y_i.$$

If the inner product of two vectors equals zero, that means that these two vectors are *orthogonal*.

Let $A$ be an $m$-by-$n$ matrix, the $(i, j)$ element of this matrix will be denoted by $a_{ij}$. The *transpose* of $A$, denoted by $A^T$, is an $n$-by-$m$ matrix resulting from swapping the roles of the row and column indices. We say that an $n$-by-$n$ matrix is *symmetric* if $A^T = A$. A symmetric matrix $A$ is said to be *positive semidefinite* if

$$x^T A x \geq 0, \qquad \text{for any} \quad x \in \mathbb{R}^n. \tag{1.4}$$

We say that $A$ is *positive definite* if strict inequality holds in (1.4), that is

$$x^T A x > 0, \qquad \text{for any} \quad x \in \mathbb{R}^n, x \neq 0.$$

Finally, a matrix $A$ is called *indefinite* if there exists $x, y \in \mathbb{R}^n$ for which

$$x^T Ax > 0 \quad \text{and} \quad y^T Ay < 0.$$

**Norms**

A *vector norm* is a function from $\mathbb{R}^n$ to $\mathbb{R}$ which has the following properties :

1. $\|x\| \geq 0$ for all $x \in \mathbb{R}^n$ and $\|x\| = 0$ if and only if $x = 0$,
2. $\|\alpha x\| = |\alpha| \|x\|$ for all $\alpha \in \mathbb{R}$ and $x \in \mathbb{R}^n$,
3. $\|x + y\| \leq \|x\| + \|y\|$ for all $x, y \in \mathbb{R}^n$.

Well-known examples of norms are $\ell_p$-norms, which are defined by

$$\|x\|_p = \left( \sum_{i=1}^{n} |x_i|^p \right)^{1/p}, \quad 1 \leq p < \infty.$$

The following norms are mainly used :

$$\|x\|_1 = \sum_{i=1}^{n} |x_i|, \tag{1.5a}$$

$$\|x\|_2 = \left( \sum_{i=1}^{n} |x_i|^2 \right)^{1/2} = \sqrt{x^T x}, \tag{1.5b}$$

$$\|x\|_\infty = \max_{1 \leq i \leq n} |x_i|. \tag{1.5c}$$

The norm defined by (1.5b) is called the $\ell_2$-norm or the *Euclidean norm*. All norms in $\mathbb{R}^n$ are *equivalent* in the sense that each one is bounded above and below by a multiple of the other. In particular, we have that

$$\|x\|_\infty \leq \|x\|_2 \leq \sqrt{n} \|x\|_\infty \quad \text{and} \quad \|x\|_\infty \leq \|x\|_1 \leq n \|x\|_\infty.$$

Note that the Euclidean norm satisfies the *Cauchy-Schwarz inequality* :

$$|x^T y| \leq \|x\|_2 \|y\|_2.$$

A *matrix norm* of an $m$-by-$n$ matrix $A$, denoted by $\|A\|$, is a function from $\mathbb{R}^{m \times n}$ to $\mathbb{R}$ such that

1. $\|A\| \geq 0$ for all $A \in \mathbb{R}^{m \times n}$ and $\|A\| = 0$ if and only if $A = 0$,
2. $\|\alpha A\| = |\alpha| \|A\|$ for all $\alpha \in \mathbb{R}$ and $A \in \mathbb{R}^{m \times n}$,

3. $\|A + B\| \leq \|A\| + \|B\|$ for all matrices $A, B \in \mathbb{R}^{m\times n}$,

4. $\|A\ B\| \leq \|A\|\ \|B\|$ for all matrices $A, B \in \mathbb{R}^{m\times n}$.

The most frequently used matrix norms in numerical analysis and optimization theory are the *Frobenius norm*

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n} |a_{ij}|^2}$$

and the $\ell_p$-norms

$$\|A\|_p = \max_{\|x\|_p=1} \|Ax\|_p = \max_{x\neq 0} \frac{\|Ax\|_p}{\|x\|_p}.$$

In particular, we have

$$\begin{aligned}
\|A\|_1 &= \max_{1\leq j\leq n} \sum_{i=1}^{m} |a_{ij}|, \\
\|A\|_2 &= \sqrt{\lambda_{max}(A^T A)}, \qquad \text{where} \quad \lambda_{max}(\cdot) \text{ denotes the largest eigenvalue}, \\
\|A\|_\infty &= \max_{1\leq i\leq m} \sum_{j=1}^{n} |a_{ij}|.
\end{aligned}$$

Consider now an $m$-by-$n$ matrix, the *null space* of $A$ is defined as the subspace

$$\text{Null } (A) \stackrel{\text{def}}{=} \{x \in \mathbb{R}^n \mid Ax = 0\}.$$

**Elements of topology**

We now briefly present some basic aspects of topology that are useful in optimization theory. Let be a point $x \in \mathbb{R}^n$, an open set containing $x$ is called a *neighbourhood* of $x$ and is denoted by $\mathcal{N}$. A frequently used neighbourhood is the open ball of radius $\varepsilon$ around $x$, which is denoted and defined by

$$\mathcal{B}_\varepsilon(x) \stackrel{\text{def}}{=} \{y \mid \|y - x\| < \varepsilon\}.$$

A subset $\mathcal{S}$ of $\mathbb{R}^n$ is said to be *open* if for each vector $x \in \mathcal{S}$ there exists a constant $\varepsilon > 0$ such that $\mathcal{B}_\varepsilon(x) \subset \mathcal{S}$. We say that a set $\mathcal{S}$ is *closed* if and only if its complement in $\mathbb{R}^n$, that is $\mathbb{R}^n \setminus \mathcal{S}$, is open. A subset $\mathcal{S}$ of $\mathbb{R}^n$ is said to be *bounded* if there exists a constant $\kappa > 0$ such that

$$\|x\| \leq \kappa \ \text{ for all } \quad x \in \mathcal{S}.$$

Finally, a set $\mathcal{S}$ of $\mathbb{R}^n$ is *compact* if and only if it is both closed and bounded.

**Convexity**

A set $\mathcal{S}$ of $\mathbb{R}^n$ is called *convex* if for any points $x, y \in \mathcal{S}$ and any $\lambda \in [0,1]$, we have that

$$\lambda x + (1-\lambda)y \in \mathcal{S}.$$

This means that all points on a line connecting two points in the set $\mathcal{S}$ are also in $\mathcal{S}$. The intersection of a finite or infinite number of convex sets is itself a convex set; but the union of convex sets is not necessarily convex. A function $f$, defined on a convex set $\mathcal{S}$, is *convex* if for all $x, y \in \mathcal{S}$, we have that

$$f(\lambda x + (1-\lambda)y) \leq \lambda f(x) + (1-\lambda) f(y) \qquad \text{for all} \quad \lambda \in [0,1]. \tag{1.7}$$

A function $f$ is called *strictly convex* if, for $x \neq y$ and $\lambda \in (0,1)$, the inequality (1.7) is strict. We say that a function $f$ is *concave* if $(-f)$ is convex. Finally, the *convex hull* of a given subset $\mathcal{S}$ of $\mathbb{R}^n$, denoted by $co\{\mathcal{S}\}$, is the intersection of all convex sets containing $\mathcal{S}$. Formally, the convex hull can be described as the set of convex combinations of points from $S$, *i.e.*

$$co\{\mathcal{S}\} = \left\{ \bar{x} \;\middle|\; \bar{x} = \sum_{i=1}^{N} \lambda_i x_i, \quad \text{where} \;\; N \in \mathbb{N}, \lambda_i \geq 0, x_i \in \mathcal{S} \;\; \text{and} \;\; \sum_{i=1}^{N} \lambda_i = 1 \right\}.$$

**Lipschitz continuity**

Let $\mathcal{S}$ be an open subset of $\mathbb{R}^n$. The function $f : \mathcal{S} \to \mathbb{R}^m$ is said to be *Lipschitz continuous* on $\mathcal{S}$ if there exists a constant $M > 0$ such that

$$\|f(x) - f(y)\| \leq M \|x - y\| \qquad \text{for all} \quad x, y \in \mathcal{S}.$$

The function $f$ is called *locally Lipschitz continuous* at a point $\bar{x} \in \text{int}\mathcal{S}$ [(1)] if there exists a neighbourhood $\mathcal{N}$ with $\bar{x} \in \mathcal{N} \subset \mathcal{S}$ and a constant $M > 0$ such that

$$\|f(\bar{x}) - f(\bar{y})\| \leq M \|\bar{x} - \bar{y}\| \qquad \text{for all} \quad \bar{x}, \bar{y} \in \mathcal{N}.$$

### 1.3.2 Generalities on convergence

We now give some generalities on the notion of convergence which is an important tool in optimization theory, in particular for qualifying the speed of convergence towards the solution.

[(1)] int$\mathcal{S}$ denotes the interior of the set $\mathcal{S}$ and is the largest open set contained in $\mathcal{S}$.

**Sequences**

A sequence $\{x_k\}(k = 1, 2, \dots)$ in $\mathbb{R}^n$ is said to be *convergent* to $x^* \in \mathbb{R}^n$ if for every $\varepsilon > 0$ there exists an index $K$ such that

$$\|x_k - x^*\| < \varepsilon \qquad \text{for all } \; k \geq K,$$

and we denote it by

$$x_k \to x^* \qquad \text{or} \qquad \lim_{k\to\infty} x_k = x^*.$$

A sequence $\{x_k\}$ is called a *bounded sequence* if there exists a constant $\kappa > 0$ such that $\|x_k\| \leq \kappa$ for all $k$. We define the *limit superior* of a sequence $\{x_k\}$ to be

$$\limsup_{k\to\infty} x_k = \inf_{k\geq 0} \sup_{K\geq k} x_K,$$

and the *limit inferior*

$$\liminf_{k\to\infty} x_k = \sup_{k\geq 0} \inf_{K\geq k} x_K.$$

We can think of the limit superior as the largest value which the sequence approaches infinitely often, and the limit inferior the smallest.

Finally, we say that $x^* \in \mathbb{R}^n$ is a *limit point* or an *accumulation point* of the sequence $\{x_k\}$ if there exists a subsequence $\{x_k\}_{k\in K} \subset \{x_k\}$ such that $\{x_k\}_{k\in K}$ converges to $x^*$.

**Rates of convergence**

The concept of rate of convergence is of practical importance if we are concerned with infinite sequences of iterates $\{x_k\}$ like in the majority of optimization processes. This is a mean of measuring the speed of convergence of the sequences.
One of the most efficient ways for assessing the rate of convergence is to make a comparison between the progress at each step and the progress at the previous step. We say that a sequence $\{x_k\}$ *converges linearly* to $x^*$ if

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|} = \mu \qquad \text{with } \; 0 < \mu < 1. \tag{1.8}$$

If (1.8) holds with the rate of convergence $\mu$ equals to zero, one says that the sequence *converges superlinearly*. More generally, a sequence $\{x_k\}$ is said to be *convergent with order r* for $r > 1$ to $x^*$ if

$$\lim_{k\to\infty} \frac{\|x_{k+1} - x^*\|}{\|x_k - x^*\|^r} = \mu \qquad \text{with } \; \mu > 0.$$

In particular, convergence with order 2 is called *quadratic convergence*. Roughly speaking, this means that the number of exact digits in $x_k$ doubles at each iteration. A detailed description of the rates of convergence of iterative sequences can be found in Ortega and Rheinboldt [104].

**Global and local convergence**

Two types of convergence are pertinent in optimization theory. The first is the *global convergence*. Assume that a sequence $\{x_k\}$ is generated by an algorithm, this latter is said to be *globally* convergent when $\{x_k\}$ converges to "what we want" for any initial iterate. By "what we want", we mean a point satisfying the necessary optimality conditions (see Section 2.2) and not necessarily a global optimum. The second is the *local convergence*. Suppose that $\{x_k\}$ has a limit point $x^*$, we wish to identify the speed of convergence of $x_k$ towards $x^*$. This study is performed by using the tools defined in the previous section.

### 1.3.3 Derivatives

This section is devoted to what is probably the most fundamental property of the functions involved in mathematical programs, namely the *differentiability*. The latter is very important because most algorithms use available information about a function at one point to deduce its behaviour at other points. If the problem derivatives are available, the capability of an algorithm to find a solution is strengthened compared with problems without derivatives.

**First and second derivatives**

Let $f : \mathbb{R} \rightarrow \mathbb{R}$ be an univariate function (that is a real-valued function of a real variable). The *first derivative* is defined by

$$f'(x) = \frac{df(x)}{dx} = \lim_{\varepsilon \rightarrow 0} \frac{f(x+\varepsilon) - f(x)}{\varepsilon}.$$

The function $f$ is said to be *differentiable* at $x$ if $f'(x)$ exists.
Consider now the multivariate function $f : \mathbb{R}^n \rightarrow \mathbb{R}$. The *first partial derivative* of $f$ with respect to $x_i$ is defined by the following limit (if it exists) :

$$\frac{\partial f(x)}{\partial x_i} = \lim_{\varepsilon \rightarrow 0} \frac{f(x+\varepsilon e_i) - f(x)}{\varepsilon},$$

where $e_i$ is the $i$th unit vector. Assuming that all of these partial derivatives exist, the function $f$ is said to be *differentiable* and the *gradient* of $f$ at $x$ is defined as the $n$-vector

$$\nabla_x f(x) = g(x) = \begin{bmatrix} \frac{\partial f(x)}{\partial x_1} \\ \vdots \\ \frac{\partial f(x)}{\partial x_n} \end{bmatrix}.$$

If $f$ is differentiable at $x$, $f$ is continuous at $x$. If the derivatives are further continuous functions of $x$, $f$ is said to be *continuously differentiable*. Assume now that $x$ depends on $y$, the derivative

of $f$ with respect to $y$ can by computed by the so-called *chain rule* :

$$\frac{df(x(y))}{dy} = \frac{df}{dx}\frac{dx}{dy}. \tag{1.9}$$

If the *second partial derivatives* of a function $f$ defined by

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} = \frac{\partial}{\partial x_i}\frac{\partial f(x)}{\partial x_j}$$

exist for all $i, j$ $(1 \leq i, j \leq n)$ and are continuous functions of $x$, then $f$ is said to be *twice-continuously differentiable*, that is $f \in C^2$. These $n^2$ second partial derivatives are represented by an $n$-by-$n$, symmetric matrix known as the *Hessian matrix* of $f$ or simply the *Hessian* :

$$\nabla^2_{xx} f(x) = \begin{pmatrix} \frac{\partial^2 f(x)}{\partial^2 x_1} & \frac{\partial^2 f(x)}{\partial x_1 \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_1 \partial x_n} \\ \frac{\partial^2 f(x)}{\partial x_2 \partial x_1} & \frac{\partial^2 f(x)}{\partial^2 x_2} & \cdots & \frac{\partial^2 f(x)}{\partial x_2 \partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f(x)}{\partial x_n \partial x_1} & \frac{\partial^2 f(x)}{\partial x_n \partial x_2} & \cdots & \frac{\partial^2 f(x)}{\partial^2 x_n} \end{pmatrix}.$$

We define the *curvature* of $f$ at $x \in \mathbb{R}^n$ along the direction $d \in \mathbb{R}^n$ by

$$\frac{d^T \nabla^2_{xx} f(x) d}{\|d\|^2}.$$

Note that if $f \in C^2$ and $d^T \nabla^2_{xx} f(x) d < 0$ $(> 0$, respectively), the vector $d$ is a *direction of negative (positive) curvature*.
A vector-valued function $f : \mathbb{R}^n \to \mathbb{R}^m$, with component functions $f_1, \ldots, f_m$ is called *differentiable* if all partial derivatives

$$\frac{\partial f_j(x)}{\partial x_i} = \lim_{\varepsilon \to 0} \frac{f_j(x + \varepsilon e_i) - f_j(x)}{\varepsilon} \qquad i = 1, \ldots, n, \quad j = 1, \ldots, m$$

exist. These partial derivatives may be gathered in an $m$-by-$n$ matrix known as the *Jacobian* of $f$ :

$$J_x f(x) = \begin{pmatrix} \frac{\partial f_1(x)}{\partial x_1} & \frac{\partial f_1(x)}{\partial x_2} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \frac{\partial f_2(x)}{\partial x_1} & \frac{\partial f_2(x)}{\partial x_2} & \cdots & \frac{\partial f_2(x)}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_m(x)}{\partial x_1} & \frac{\partial f_m(x)}{\partial x_2} & \cdots & \frac{\partial f_m(x)}{\partial x_n} \end{pmatrix} = \begin{pmatrix} \nabla_x f_1(x) \\ \vdots \\ \nabla_x f_m(x) \end{pmatrix}.$$

**Taylor approximations**

One of the results from analysis that is most frequently used in optimization theory is the following theorem.

**Theorem 1.1** (Mean value theorem)
Suppose that $f : \mathbb{R}^n \to \mathbb{R}$ is continuously differentiable and that $s \in \mathbb{R}^n$. Then we have that

$$f(x+s) = f(x) + \nabla_x f(x+\alpha s)^T s,$$

for some $\alpha \in (0,1)$. If, in addition, $f$ is twice-continuously differentiable, we have that

$$\nabla_x f(x+s) = \nabla_x f(x) + \int_0^1 \nabla^2_{xx} f(x+\alpha s) s \, d\alpha,$$

and that

$$f(x+s) = f(x) + \nabla_x f(x)^T s + \frac{1}{2} s^T \nabla^2_{xx} f(x+\alpha s) s$$

for some $\alpha \in (0,1)$.

This theorem shows that, if the function and its first and second derivatives are known at a point $x$, then we can build approximations to that function at all points in the neighbourhood of $x$. In particular, we may approximate $f(x+s)$ by its *first-order Taylor approximation*

$$f(x+s) \approx f(x) + \nabla_x f(x)^T s \tag{1.10}$$

or by its *second-order Taylor approximation*

$$f(x+s) \approx f(x) + \nabla_x f(x)^T s + \tfrac{1}{2} s^T \nabla^2_{xx} f(x) s. \tag{1.11}$$

Note that equations (1.10) and (1.11) are also respectively called linear and quadratic *models* of the function $f$ around the point $x$ and are widely used in algorithms designed for solving optimization problems.

### 1.3.4 Approximation to derivatives

In many mathematical programs, it may happen that the derivatives are not available in an explicit form or that they are formulated in a very complicated way. A common way to approximate first and second derivatives of a function is to use finite-difference approximations. The technique of finite differencing is inspired by Theorem 1.1. In fact, the derivatives are a measure of the sensitivity of the function to infinitesimal changes in the values of the variables. Note that this approach is valid only in the smooth case. Approximations to first and second derivatives are current in optimization algorithms and complete discussions on this topic may be found, for instance, in the books of Dennis and Schnabel [41], Gill, Murray and Wright [62] and Nocedal and Wright [101] or in the paper of Gill et al. [61].

**Finite-difference gradients**

The first derivative $\nabla_x f(x)$ of a function $f$ may be approximated componentwise by the *forward finite-difference approximation* of the gradient

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + h_i e_i) - f(x)}{h_i}, \quad i = 1, \ldots, n, \tag{1.12}$$

where $h \in \mathbb{R}^n$ is a vector of stepsizes and $e_i$ is the $i$th unit vector. A more accurate approximation to the gradient can be obtained by using the *central finite-difference approximation* given by

$$\frac{\partial f(x)}{\partial x_i} \approx \frac{f(x + h_i e_i) - f(x - h_i e_i)}{2h_i}, \quad i = 1, \ldots, n. \tag{1.13}$$

An important issue in the implementation of these formulae is the choice of the stepsize $h$. Note that the forward approximation requires $n$ additional function evaluations while the central one requires $2n$.

**Finite-difference Hessians**

Finite-difference approximations are also possible for second-order derivative matrices. We may consider two ways to estimate the Hessian $\nabla^2_{xx} f(x)$ by finite differences, one when the first derivative $\nabla_x f(x)$ is analytically available and the other one when it is not.
If the gradient of the objective function is analytically available, the Hessian matrix can be obtained by applying the *forward finite-difference* formula

$$\frac{\partial(\nabla_x f(x))}{\partial x_i} \approx \frac{\nabla_x f(x + h_i e_i) - \nabla_x f(x)}{h_i}, \quad 1 \leq i \leq n, \tag{1.14}$$

or the *central finite-difference* one

$$\frac{\partial(\nabla_x f(x))}{\partial x_i} \approx \frac{\nabla_x f(x + h_i e_i) - \nabla_x f(x - h_i e_i)}{2h_i}, \quad 1 \leq i \leq n. \tag{1.15}$$

Remark that this column-at-a-time process does not necessarily lead to a symmetric matrix. Nevertheless, denoting the approximation to $\nabla^2_{xx} f(x)$ by $B$, we can restore the symmetry by applying $B = (B + B^T)/2$. The forward formula requires $n$ additional gradient calls while, for the central one, $2n$ additional gradient evaluations are needed.
Consider now the case in which even first derivatives are not available. We can use the following finite-difference formula that uses only function values

$$\frac{\partial^2 f(x)}{\partial x_i \partial x_j} \approx \frac{f(x + h_i e_i + h_j e_j) - f(x + h_i e_i) - f(x + h_j e_j) + f(x)}{h_i h_j}, \tag{1.16}$$

for $1 \leq i \leq j \leq n$. Taking symmetry into account, this approximation requires $\frac{1}{2}(n^2 + 3n)$ additional function evaluations to that of $f(x)$. Obviously, this is relatively expensive and therefore, in practice, this strategy is used only if the cost of a function evaluation is not too high. So, in the situation where the first and *a fortiori* the second derivatives of the function involved in the optimization problem are not available or are time-consuming, we often prefer to consider *derivative-free optimization* techniques (see, for instance, Conn, Scheinberg and Toint [35, 36]).

For the selection of the stepsize $h$ in all those finite-difference formulae, we have to make a compromise between large rounding errors, due to small stepsizes, and large approximation errors, generated by large stepsizes. Note that possible values for $h$ will be discussed in Chapter 6 but a typical one is $h_i = \sqrt{\varepsilon_M}$ for (1.12) and $h_i = \sqrt[3]{\varepsilon_M}$ for (1.13), where $\varepsilon_M$ is the machine precision.

### 1.3.5 Automatic differentiation

Another conceivable approach to treat with mathematical programs involving unavailable derivatives is the *automatic differentiation*, also referred to as the *computational differentiation*. This is a set of techniques using the computational representation of a function and based on the application of the chain rule (1.9) to construct exact values for the derivatives. Automatic differentiation techniques exploit the fact that every function executes a sequence of simple elementary arithmetic operations involving at most two arguments at a time, such as addition, multiplication, division, the power operation or exponential, logarithmic and trigonometric functions. By processing the chain rule repeatedly to these operations, derivatives of arbitrary order can be obtained automatically. Note that such techniques do not need to know anything about the problem. There exist two principal techniques of automatic differentiation: the forward and the reverse modes. The reader should refer to Griewank and Corliss [79] and Griewank [77, 78] for an introduction to this topic.

# Chapter 2

# Background on nonlinear optimization

This work principally addresses the solution of nonlinear optimization problems, also called nonlinear programming problems (NLP). A nonlinear program consists of the optimization of an objective function possibly subject to some constraint functions, where any of these functions may be nonlinear. These kinds of problems can arise from science and engineering. For instance, the energy dissipated in an electric circuit is a nonlinear function of the resistances; the volume of a sphere is a nonlinear function of its radius.

In this chapter we present a brief review of some existing methods for solving optimization problems. We first give some characterizations of solutions of optimization problems by way of optimality conditions. We next present some well-known algorithms for unconstrained and constrained nonlinear programming as these are at the root of methods developed in the framework of this thesis. However, this survey is far from being exhaustive; we do not, for example, discuss interior-point methods (see Wright [129]).

## 2.1 Characterization of solutions

Consider the general mathematical program

$$\min_{x \in \mathcal{S}} f(x), \tag{2.1}$$

where $f$ is a function from $\mathbb{R}^n$ into $\mathbb{R}$ and the feasible set $\mathcal{S}$, is a subset of $\mathbb{R}^n$. Before discussing methods and algorithms for solving nonlinear programming problems, we must characterize a "solution" of problem (2.1). The first thing to know is that only feasible points may be optimal. A point $x^*$ is a *global minimizer* of $f$ if

$$f(x^*) \leq f(x) \quad \text{for all} \quad x \in \mathcal{S}. \tag{2.2}$$

Nevertheless, finding the global minimizer of a problem can be a very complicated task. This is why most practical nonlinear optimization algorithms only aim at finding a local minimizer, which is a point that has the lowest value of $f$ in its neighbourhood. More formally, we say that a point $x^*$ is a *(weak) local minimizer* if there is a neighbourhood $\mathcal{N}$ of $x^*$ such that

$$f(x^*) \leq f(x) \text{ for all } x \in \mathcal{N} \cap \mathcal{S}.$$

A *strict local minimizer* is a point $x^*$ if there exists a neighbourhood $\mathcal{N}$ of $x^*$ such that

$$f(x^*) < f(x) \text{ for all } x \in \mathcal{N} \cap \mathcal{S} \quad \text{with} \quad x \neq x^*.$$

We call *minimum* the value of the objective function $f$ at a minimizer. The following figure illustrates the types of minimizers defined above.

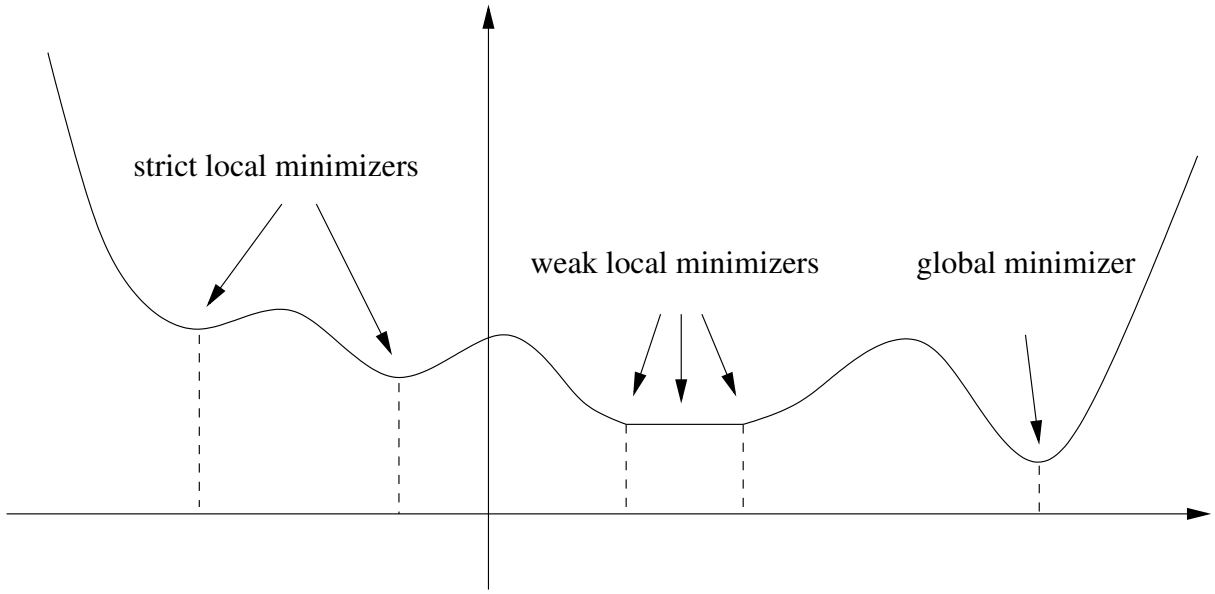


Figure 2.1: Examples of local and global minimizers in one dimension

Note that, in this work, we restrict our attention to the computation of local minimizers.

## 2.2 Optimality conditions

This section will be concerned with *optimality conditions* which are the mathematical tools used to verify whether or not a given point is an optimal solution of the problem under study. These conditions verify if a point $x^*$ has the smallest function value in its neighbourhood.

Moreover, the information given by the optimality test is often the basis for the computation of the next trial point. In the two following sections, we state the classical optimality conditions for general unconstrained and constrained optimization problems. Further details about optimality conditions may be found, for instance, in the books of Bazaraa et al. [4], Conn et al. [34] and Nocedal and Wright [101].

### 2.2.1 Unconstrained optimization

Let us consider the unconstrained minimization problem

$$\min_{x \in \mathbb{R}^n} \; f(x), \tag{2.3}$$

we begin with necessary conditions for optimality. These are deduced by assuming that a point $x^*$ is a local minimizer and then establishing some properties of the first-order and second-order derivatives.

**Theorem 2.1** (First-order necessary condition)
Suppose that $x^*$ is a local minimizer of problem (2.3) and $f$ is continuously differentiable in an open neighbourhood of $x^*$. Then

$$\nabla_x f(x^*) = 0. \tag{2.4}$$

A point $x^*$ satisfying (2.4) is referred to as a *first-order critical* or *first-order stationary point* of $f$. Note that any local minimizer must be a first-order critical point.

**Theorem 2.2** (Second-order necessary conditions)
Suppose that $x^*$ is a local minimizer of problem (2.3) and $f$ is twice-continuously differentiable in an open neighbourhood of $x^*$. Then

$$\nabla_x f(x^*) = 0 \quad \text{and} \quad \nabla^2_{xx} f(x^*) \;\; \text{is positive semidefinite.}$$

In this case, we say that $x^*$ is a *second-order critical point*. The following theorem states under what conditions a point $x^*$ is a local minimizer of the objective function $f$.

**Theorem 2.3** (Second-order sufficient conditions)
Suppose that $f$ is twice-continuously differentiable in an open neighbourhood of $x^*$ and that furthermore

$$\nabla_x f(x^*) = 0 \quad \text{and} \quad \nabla^2_{xx} f(x^*) \text{ is positive definite.}$$

Then $x^*$ is a strict local minimizer of problem (2.3).

Note that, if $x^*$ is first-order critical but the Hessian is indefinite, we say that $x^*$ is a *saddle point*.

In the context of convex programming (see Section 1.2.2), optimality conditions are simpler. Indeed, if the objective function $f$ is convex, every local minimizer of this function is also a global minimizer. This leads to the following global optimality condition that is both necessary and sufficient.

**Theorem 2.4** (Convex programming)
Assume that the objective function $f$ is convex and continuously differentiable. Then $x^*$ is a global minimizer of (2.3) if and only if $\nabla_x f(x^*) = 0$.

Finally, we point out the fact that optimality conditions often provide the foundations for the development and the analysis of iterative algorithms. In the case of unconstrained optimization, algorithms search for points at which the gradient of $f$ vanishes, as our filter-trust-region algorithm described in Chapter 4. In practice, algorithms terminate when these optimality conditions hold approximately.

### 2.2.2 Constrained optimization

Consider now a general constrained problem of the following form

$$\begin{array}{ll} \text{minimize} & f(x) \\ \text{subject to} & x \in X, \\ & c_i(x) = 0, \quad i \in \mathcal{E}, \\ & c_i(x) \leq 0, \quad i \in \mathcal{I}. \end{array} \tag{2.5}$$

Firstly, we define the *active set* at any feasible point $x$ as the set of indices of the constraints

at which equality holds, that is

$$\mathcal{A}(x) = \{i \in \mathcal{E} \cup \mathcal{I} \mid c_i(x) = 0\}.$$

We say that the constraints whose indices belong to the active set are *active*. If $j \notin \mathcal{A}(x)$, we say that the $j$th constraint is *inactive* at $x$.
The *Lagrangian* (or *Lagrange function*) for problem (2.5) is defined by

$$\mathcal{L}(x, \lambda) = f(x) + \sum_{i \in \mathcal{E} \cup \mathcal{I}} \lambda_i \, c_i(x),$$

where $\lambda_i, i \in \mathcal{E} \cup \mathcal{I}$, are known as the *Lagrange multipliers*. Usually, the components of $x$ are called the *primal variables* and the Lagrange multipliers are known as the *dual variables*.

Assuming some properties of the constraint functions, known as *constraint qualifications*, necessary and sufficient conditions for local minima can be established. These constraint qualification conditions ensure that pathological behaviours do not occur at $x^*$ while analysing solutions of constrained mathematical programs. They guarantee that, near to a presumed minimum, any nonlinear constraint is reasonably approximated by its first-order Taylor approximation (1.10). We point out the fact that this constraint qualification is relevant only to nonlinear constraints. One of the most frequently used constraint qualification at a local solution $x^*$ is the following.

**Condition 2.1** (Linear independence constraint qualification (LICQ))
Given a point $x^*$ and its corresponding active set $\mathcal{A}(x^*)$, the *linear independence constraint qualification* (LICQ) holds for problem (2.5) at $x^*$ if the set of the gradients of active constraints at $x^*$

$$\{\nabla_x c_i(x^*) : \; i \in \mathcal{A}(x^*)\} \tag{2.6}$$

are linearly independent.

Another useful constraint qualification is established below.

**Condition 2.2** (Slater's constraint qualification)
Assuming the equality constraints $c_i(x), i \in \mathcal{E}$ are affine and the inequality constraints $c_i(x), i \in \mathcal{I}$ and the objective function $f$ are convex, the *Slater's constraint qualification* holds if the gradients $\nabla_x c_i(x), i \in \mathcal{E}$ are linearly independent vectors and there exists $\tilde{x} \in \mathbb{R}^n$ such that $c_i(\tilde{x}) < 0$ for all $i \in \mathcal{I}$ and $c_i(\tilde{x}) = 0$ for all $i \in \mathcal{E}$.

There are other constraint qualifications, such as the Mangasarian-Fromovitz constraint qualification (MFCQ) which is a weaker condition than LICQ. For a more complete description of the MFCQ and other constraint qualifications, we refer the reader to the book of Nocedal and Wright [101]. With the LICQ condition, we are now able to state the *first-order necessary conditions*.

**Theorem 2.5** (First-order necessary conditions)
Let $x^*$ be a local solution of problem (2.5) at which the LICQ condition holds. Then there exists a Lagrange multiplier vector $\lambda^*$, with components $\lambda_i^*$ ($i \in \mathcal{E} \cup \mathcal{I}$), such that the following conditions are satisfied at $(x^*, \lambda^*)$:

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0, \tag{2.7a}$$
$$c_i(x^*) = 0 \quad \text{for all} \quad i \in \mathcal{E}, \tag{2.7b}$$
$$c_i(x^*) \leq 0 \quad \text{for all} \quad i \in \mathcal{I}, \tag{2.7c}$$
$$\lambda_i^* \geq 0 \quad \text{for all} \quad i \in \mathcal{I}, \tag{2.7d}$$
$$\lambda_i^* c_i(x^*) = 0 \quad \text{for all} \quad i \in \mathcal{E} \cup \mathcal{I}. \tag{2.7e}$$

These conditions are also known as the *Karush-Kuhn-Tucker (KKT) conditions*. The necessary conditions for equality and inequality constrained problems were first published in Karush's Master's thesis [85], and they were re-introduced by Kuhn and Tucker in [86]. The condition (2.7a) is often referred as the *stationary condition*, while conditions (2.7b) and (2.7c) are called *(primal) feasibility conditions*. Finally, the last equation (2.7e) is the *complementarity slackness condition*. A point $x^*$ satisfying the KKT conditions is called a *KKT point* or a *first-order critical point* for problem (2.5). As the condition (2.7e) implies that the Lagrange multipliers associated to inactive inequality constraints are zero, we can rewrite the condition (2.7a) as

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = \nabla_x f(x^*) + \sum_{i \in \mathcal{A}(x^*)} \lambda_i^* \nabla_x c_i(x^*) = 0.$$

As in the unconstrained case, we consider the particular case of convex programming. If the Slater's constraint qualification (2.2) holds, the KKT conditions are both necessary and sufficient for optimality. We can derive the following theorem.

**Theorem 2.6** (Convex programming)
If $f$ is convex and the feasible region is convex, any local solution of the constrained problem (2.5) is also a global solution.

Further, if $f$ and $c$ are differentiable and if Slater's condition (2.2) holds, (2.7a)-(2.7e) are *necessary and sufficient* conditions for $x^*$ (and $\lambda^*$) to define a solution.
Furthermore, if $f$ is *strictly convex*, the global solution is *unique*.

Using second-order derivatives information, we now establish the second-order optimality conditions for the general constrained problem (2.5). In order to introduce these conditions, we must give the following set definition where $x \in \mathbb{R}^n$ and $\lambda \in \mathbb{R}^{p+m}$ are given vectors, with $p = \#\mathcal{I}$ and $m = \#\mathcal{E}$,

$$\mathcal{N}_+(x,\lambda) = \left\{ s \in \mathbb{R}^n \;\middle|\; \begin{array}{l} s^T \nabla_x c_i(x) = 0 \quad \forall i \in \mathcal{E} \cup \{j \in \mathcal{A}(x) \cap \mathcal{I} : \lambda_j > 0\} \\ \text{and } s^T \nabla_x c_i(x) \leq 0 \quad \forall i \in \{j \in \mathcal{A}(x) \cap \mathcal{I} : \lambda_j = 0\} \end{array} \right\}.$$

**Theorem 2.7** (Second-order necessary conditions)
Let $x^*$ be a local solution of problem (2.5) at which the LICQ condition holds. Let also $\lambda^*$ be a Lagrange multiplier vector such that the KKT conditions (2.7a)-(2.7e) are satisfied. Then

$$s^T \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) s \geq 0 \quad \text{for all } s \in \mathcal{N}_+(x^*, \lambda^*). \tag{2.8}$$

Property (2.8) states that the curvature of the Lagrangian along directions in $\mathcal{N}_+(x^*, \lambda^*)$ must be nonnegative. We say that a point $x^*$ satisfying (2.8) is a *strong second-order critical point*.
It is possible to formulate a converse to Theorem 2.7 and derive the following second-order sufficient conditions for optimality.

**Theorem 2.8** (Second-order sufficient conditions)
Suppose that for some feasible point $x^* \in \mathbb{R}^n$ there exists a Lagrange multiplier vector $\lambda^*$ satisfying the KKT conditions (2.7a)-(2.7e). Assume also that

$$s^T \nabla^2_{xx} \mathcal{L}(x^*, \lambda^*) s > 0 \quad \text{for all } s \in \mathcal{N}_+(x^*, \lambda^*) \text{ with } s \neq 0.$$

Then $x^*$ is a strict local solution of problem (2.5).

### 2.2.3 Criticality measure

In our later developments (see in particular Section 7.2), we will need the concept of *criticality measure*. We define $\pi(k, x_k)$ to be a *first-order criticality measure* of the iterate $x_k$ if it is a nonnegative real function of its second argument such that

$$\|x_k - x_\ell\| \to 0 \quad \text{implies that} \quad |\pi(k, x_k) - \pi(\ell, x_\ell)| \to 0$$

and if the limit

$$\lim_{k\to\infty} \pi(k, x_k) = 0$$

corresponds to asymptotically satisfying the first-order criticality conditions of the optimization problem under study (see Conn et al. [34, Section 8.1] for further details).

## 2.3 Methods for nonlinear unconstrained optimization

We start our review with algorithms designed for solving unconstrained nonlinear optimization problems, that is mathematical programs of the form (2.3). As we have already pointed out in Section 2.2.1, unconstrained optimization algorithms search for points at which the gradient of the objective function $f$ vanishes. So a simple idea for solving this kind of problems is to try to solve the system of $n$ equations with $n$ unknowns given by

$$\nabla_x f(x) = 0. \tag{2.9}$$

Usually, an *iterative* method is needed for solving (2.9). Such a method iteratively produces a sequence of iterates

$$x_1, x_2, \ldots, x_k, \ldots$$

hoping that this sequence converges to a solution of the problem (2.3). In order to compute the next iterate, iterative algorithms need to know some information about the problem. First of all, they ought to know the value of the objective function $f$ for every $x$. Sometimes, the derivatives are also required. Methods using only first-order information, like the steepest-descent method, are not very powerful. Frequently, efficient optimization algorithms must take second-order information into account.

### 2.3.1 Local methods

In this section we describe some of the best-known methods for unconstrained nonlinear optimization. These methods have only local convergence properties, we will see in Sections 2.3.2 and 2.3.3 that two kinds of *globalization techniques* may be used to enforce the convergence to a local minimum when started far away from all local minima.

**Newton's method**

We now describe Newton's method, which is an iterative one and whose aim is to solve equation (2.9). In order to derive Newton's method in the variant used in optimization, we consider the quadratic model of $f$ at the current iterate $x_k$

$$m_k(x_k + s) = f(x_k) + \nabla_x f(x_k)^T s + \frac{1}{2} s^T \nabla^2_{xx} f(x_k) s. \tag{2.10}$$

The idea is then to minimize the model at the current iterate. If the Hessian $\nabla^2_{xx} f(x_k)$ is positive definite , then the quadratic model (2.10) has a unique minimizer $s_k$ at a point where the gradient of this model vanishes, *i.e.* where

$$\nabla_x f(x_k) + \nabla^2_{xx} f(x_k) s = 0.$$

As long as $\nabla^2_{xx} f(x)$ remains nonsingular for all $x$, Newton's method is well defined. It is summarized by the following algorithm.

**Algorithm 2.1: Newton's method**

**Step 0.** An initial point $x_0 \in \mathbb{R}^n$ is given. Set $k = 0$.

**Step 1.** Compute the step $s_k$ by solving the system of linear equations

$$\nabla^2_{xx} f(x_k) s_k = -\nabla_x f(x_k) \tag{2.11}$$

**Step 2.** Set

$$x_{k+1} = x_k + s_k.$$

Increment $k$ by one and go to Step1.

Equations (2.11) are called *Newton equations* and the solution $s_k$ is known as the *Newton step* or *Newton direction*. If the Hessian is positive definite, it then follows from (2.11) that

$$\nabla_x f(x_k)^T s_k = -s_k^T \nabla^2_{xx} f(x_k)\, s_k \leq -\sigma_k \|s_k\|^2 \quad \text{for some } \sigma_k > 0.$$

Therefore, if the gradient of $f$ is nonzero, the Newton direction is a *descent direction*, *i.e.*, $s_k^T \nabla_x f(x_k) < 0$. If the gradient is zero, then the step $s_k$ is also zero. However, far from a local minimum, the Hessian matrix may be singular or the Newton direction may not be a descent direction if $\nabla^2_{xx} f(x_k)$ is not positive definite. For example, if the Hessian is not

positive definite when the iterate is far from the solution, the sequence of generated iterates may converge towards a maximum or a saddle point of $f$, since any first-order critical point is a solution of the system (2.9).

Another important characteristic of Newton's method is that, when it works, it rapidly converges; the asymptotic rate of convergence is quadratic. Note the fact that Newton's method converges after one iteration for all quadratic objective functions. However, one of its most serious disadvantages is the lack of global convergence. As the Taylor's approximation (2.10) is only valid in the proximity of the solution $x^*$, Newton's method is suitable only when the initial point is close enough to $x^*$. The requirement of analytic second-order derivatives of the objective function is another drawback to Newton's method. These derivatives may be difficult to determine notwithstanding the fact that they are known to exist.
Finally, we say that the advantages and disadvantages of Newton's method are the foundations to the development and improvement of more practical algorithms.

**Quasi-Newton methods**

We now focus on Quasi-Newton methods, which, in opposition to Newton's method, do not require the computation of the exact Hessian. The basic idea of these techniques is to update approximations of the Hessian matrices in some computational cheap ways. They use the observed behaviour of the objective function and its gradient to build up curvature information to create an approximation to the Hessian using a suitable updating technique. One of their major advantages is, of course, that they do not require computation of second derivatives, neither additional function or gradient evaluations. We refer the reader to the monograph of Dennis and Schnabel [41, Chapter 9] for an introduction to this topic.

We consider here a quadratic model of the objective function $f$ at the $k$th iterate, that is

$$m_k(x_k + d) = f(x_k) + d^T \nabla_x f(x_k) + \frac{1}{2} d^T B_k d, \tag{2.12}$$

where $B_k$ is an $n$-by-$n$ symmetric positive definite matrix which approximates the matrix $\nabla^2_{xx} f(x_k)$. As this model is convex, the step which minimizes (2.12) is

$$d_k = -B_k^{-1} \nabla_x f(x_k).$$

Once the next iterate $x_{k+1}$ has been computed, a new Hessian approximation $B_{k+1}$ is obtained by updating $B_k$ to take into account information obtained during the most recent step. An *updating formula* is of the form

$$B_{k+1} = B_k + U_k,$$

where $U_k$ is called the *updating matrix*. This update represents a trial to enhance $B_k$ with information gained on the $k$th iteration.
Considering the Taylor approximation of the gradient of the objective function around the new iterate $x_{k+1}$, we obtain that the Hessian matrix $\nabla^2_{xx} f(x_k)$ verifies

$$\nabla^2_{xx} f(x_k)(x_{k+1} - x_k) \approx \nabla_x f(x_{k+1}) - \nabla_x f(x_k).$$

Therefore, we require that the approximation $B_{k+1}$ satisfies

$$B_{k+1} s_k = y_k, \tag{2.13}$$

where $s_k = x_{k+1} - x_k$ and $y_k = \nabla_x f(x_{k+1}) - \nabla_x f(x_k)$. Equation (2.13) is called the *Quasi-Newton equation* or the *secant equation*. This is why Quasi-Newton methods are sometimes called *secant methods*.

One of the first updating formula was suggested by Davidon [38] in 1959. His formula has the property of preserving positive definiteness and was later popularized by Fletcher and Powell [56] in 1963. It is known as the *DFP updating formula* which is given by

$$B_{k+1} = \left(I - \frac{y_k s_k^T}{y_k^T s_k}\right) B_k \left(I - \frac{s_k y_k^T}{y_k^T s_k}\right) + \frac{y_k y_k^T}{y_k^T s_k}.$$

It is to note that the DFP formula with exact line search (see Section 2.3.2) works well in practice.

We now present probably the most effective and widespread Quasi-Newton updating formula, namely the *BFGS formula*, which was discovered independently by Broyden [15], Fletcher [47], Goldfarb [63] and Shanno [116] in 1970. The Hessian update can be calculated by

$$B_{k+1} = B_k - \frac{B_k s_k s_k^T B_k}{s_k^T B_k s_k} + \frac{y_k y_k^T}{y_k^T s_k}. \tag{2.14}$$

Note that if $B_k$ is positive definite then the update $B_{k+1}$ will also be positive definite (when $y_k^T s_k > 0$). Moreover, the BFGS updating formula has efficient self-correcting properties. Indeed, if the matrix $B_k$ erroneously approximates the curvature in the objective function $f$, and if this slows down the iterative process, then the BFGS update tends to correct this Hessian approximation within a few steps (see Nocedal and Wright [101]).
In some situations, the updating formula can produce bad results, for example, when $y_k^T s_k$ is negative or too close to zero, we can however simply skip the Hessian update and set $B_{k+1} = B_k$. We also skip the update if $y_k$ is sufficiently close to $B_k s_k$.

Contrary to the BFGS technique, which is a rank-two updating formula, the SR1 formula is a *symmetric-rank-one* update; and it does not guarantee to produce a positive definite matrix.

However, the SR1 updating formula combined with a trust-region method (see Section 2.3.3) has proved to be quite useful (see Conn, Gould and Toint [30] or Byrd, Khalfan and Schnabel [18]). This is an update which allows indefinite approximations. The SR1 update of the Hessian matrix is given by

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}. \tag{2.15}$$

One of the drawbacks to this method is that the denominator $(y_k - B_k s_k)^T s_k$ can vanish or become very small; so, in this case, we skip the update and set $B_{k+1} = B_k$.

We also have to discuss the choice of the initial Hessian approximation $B_0$. There is no magic formula to determine it. It is common to start Quasi-Newton updates with an initial approximation $B_0$ set to the identity matrix. But other choices, that we list below, are conceivable :

- a factor of the identity matrix, for example, $B_0 = |f(x_0)|\ I$ ;
- $B_0$ may also be rescaled before $B_1$ is computed as $B_0 = \frac{y_0^T s_0}{s_0^T B_0 s_0} I$ (for more detail see Shanno and Phua [117]);
- some finite-difference approximation at $x_0$.

Note that, there is no guarantee that the finite-difference approximation to $\nabla^2_{xx} f(x)$ is positive definite. If it is not the case, one could correct the matrix into a positive definite one by using the techniques described in Section 5.5 of Dennis and Schnabel [41].

We can point out the fact that these Quasi-Newton methods are not directly applicable to large-scale optimization problems since their Hessian approximations are generally dense. Therefore, one of the major disadvantages of these methods is the large storage requirement. To circumvent this, Nocedal [100] derived a technique that considerably reduced the storage issue caused by the BFGS update. So when treating large problems, it is preferable to use *limited memory BFGS update*. For a complete description of this technique, we refer the reader to Liu and Nocedal [90] and Byrd et al. [19].

Having introduced these techniques, we will present in the next section the leading ideas of two globalization techniques, namely *line-search methods* and *trust-region methods*. These approaches are applied in order to enforce algorithms to converge from any initial point. When we are far away from a solution, these globalization techniques are an active part of the process because they avoid displacement away from the solution or even divergence. But close to an optimum, they will play the role of safeguards; they may be used if required, but usually they will

not be invoked. We particularly focus on trust-region methods since these are the foundations for our algorithmic developments in Chapters 4 and 7.

### 2.3.2 Line-search methods

We start our survey of globalization techniques with line-search methods. After having computed a direction $d_k$, each iteration of a line-search method must decide how long the step in this direction should be. The iterates are generated by

$$x_{k+1} = x_k + \alpha_k d_k,$$

where $d_k \in \mathbb{R}^n$ is the direction and $\alpha_k > 0$ is known as the *step length*, computed by an appropriate line-search method, whose goal is to *sufficiently* reduced the value of the objective function $f$. The best choice for the step length $\alpha_k$ is theoretically the solution of the following minimization problem

$$\min_{\alpha>0} f(x_k + \alpha d_k). \tag{2.16}$$

However, the exact minimizer of (2.16) is often expensive to compute and unnecessary. Therefore, instead of solving (2.16) exactly and performing an *exact line search*, it is generally preferable to solve this problem only approximately, this process is known as *inexact line search*. There are several rules for choosing the step length $\alpha$. In practice, one solves this problem approximately by imposing some conditions ensuring a sufficient decrease, as, for instance, *Armijo conditions* or *Wolfe conditions* (see Nocedal and Wright [101]). Note that, in order to benefit from the properties of fast convergence of Newton-type methods, we always try the unit step length $\alpha = 1$ first. If it is not possible, a *backtracking procedure* is performed. The value resulting from this procedure is generated by moving backwards from $\alpha = 1$ to $\alpha = 0$.

Further details about strategies and convergence analysis of line searches may be found in the monographs of Dennis and Schnabel [41] and Nocedal and Wright [101].

### 2.3.3 Trust-region methods

We now consider the second broad class of globalization techniques, called *trust regions*, which is the approach we consider in our research work to promote global convergence. The first methods that we might consider as trust-region ones are due to Levenberg [87] and Marquardt [92] and are used to solve nonlinear least-squares problems. Since then they have been extended to more general optimization problems. While line-search methods start by finding a direction and then compute the step length, trust-region algorithms first choose a maximum distance, the trust-region radius, before computing a direction and a step length simultaneously. The basic idea of trust-region methods is to accept the minimum of a quadratic model only

as long as the latter adequately reflects the behaviour of the objective function $f$, in a manner that is now well established (see Conn, Gould and Toint [34] for an extensive description of trust-region methods and their properties). In contrast to line-search techniques, trust-region methods do not require the positive definiteness of the Hessian of the quadratic model. Another interesting feature of trust-region methods is that they have the possibility to naturally take advantage of directions of negative curvature when there are present. Indeed, these directions may be taken safely to the boundary of the trust region.

At each iteration $k$, we built a model of the objective function $f$ around the current iterate $x_k$ which is easier to handle than $f$. The most practical choice for this model is a quadratic one of the form

$$m_k(x_k + s) = f(x_k) + g_k^T s + \frac{1}{2} s^T H_k s, \tag{2.17}$$

where

$$g_k \stackrel{\text{def}}{=} \nabla_x m_k(x_k) = \nabla_x f(x_k),$$

and $H_k$ is either the Hessian $\nabla^2_{xx} f(x_k)$ or some symmetric approximation to it. The purpose of trust-region methods is to restrict the search to a neighbourhood of the iterate $x_k$, in which we believe the model to be sufficiently adequate. Such a vicinity, named a *trust region*, may be defined as

$$\mathcal{B}_k = \{x_k + s \mid \|s\|_k \leq \Delta_k\},$$

where $\Delta_k > 0$ is known as the *trust-region radius* and $\|\cdot\|_k$ is an iteration-dependent norm. A classical choice is the Euclidean norm but other picks are possible according to the geometry of the problem under study. Throughout this work, we treat the Euclidean norm unless specified. Using the trust region we define the following subproblem

$$\begin{array}{ll} \min & m_k(x_k + s) \\ \text{s.t.} & \|s\| \leq \Delta_k, \end{array} \tag{2.18}$$

which is called the *trust-region subproblem*. A trial step $s_k$ is then computed by minimizing (2.18) (possibly only approximately). Let us denote the candidate point by $x_k^+ = x_k + s_k$. Trust-region algorithms then evaluate the objective function at the candidate point and accept $x_k^+$ as the new iterate if the reduction achieved in the objective function is at least a fraction of that predicted by the model. The trust-region radius $\Delta_k$ is also possibly enlarged if this is the case. This iteration is then declared to be *successful*. The increase may indicate that longer steps would be successful as well. Otherwise, if the achieved reduction is too small, the trial point is rejected, the trust-region radius is reduced and the iteration is said to be *unsuccessful*. We then conclude that the model is not sufficiently accurate in this trust region.
Formally, we compute the ratio between the *achieved reduction* (*i.e.* the decrease in $f$) versus

the *predicted reduction* (*i.e.* the reduction in $m_k$)

$$\rho_k = \frac{f(x_k) - f(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}. \tag{2.19}$$

If this ratio is close to one, it means that the model approximates the objective function well.

The detailed trust-region algorithm, which is one of the basic components of methods developed in this research work, is outlined in Algorithm 2.2.

**Algorithm 2.2: Basic Trust-Region (BTR) algorithm**

**Step 0: Initialization.** An initial point $x_0$ and an initial trust-region radius $\Delta_0 > 0$ are given. The constants $\eta_1, \eta_2, \gamma_1$ and $\gamma_2$ also given and satisfy

$$0 < \eta_1 \leq \eta_2 < 1 \quad \text{and} \quad 0 < \gamma_1 \leq \gamma_2 < 1. \tag{2.20}$$

Compute $f(x_0)$ and set the iteration counter $k$ to 0.

**Step 1: Model definition.** Compute the model $m_k$ approximating $f$ around $x_k$ in $\mathcal{B}_k$.

**Step 2: Step calculation.** Compute a step $s_k$ that "sufficiently reduces" the model $m_k$ and such that $x_k + s_k \in \mathcal{B}_k$.

**Step 3: Acceptance of the trial point.** Compute $f(x_k + s_k)$ and define $\rho_k$ as in (2.19). If $\rho_k \geq \eta_1$, set $x_{k+1} = x_k + s_k$; otherwise define $x_{k+1} = x_k$.

**Step 4: Trust-region radius update.** Update the radius as follows

$$\Delta_{k+1} \in \begin{cases} [\Delta_k, \infty) & \text{if } \rho_k \geq \eta_2 \\ [\gamma_2 \Delta_k, \Delta_k] & \text{if } \rho_k \in [\eta_1, \eta_2) \\ [\gamma_1 \Delta_k, \gamma_2 \Delta_k] & \text{if } \rho_k < \eta_1, \end{cases}$$

set $k = k + 1$ and go to Step 1.

One of the critical tasks in such an algorithm is the computation of the trial step $s_k$ which "sufficiently reduces" the model within the current trust region. This sufficient decrease may be measured in terms of the Cauchy point. Consider the *Cauchy arc*

$$x_k^C(t) = \{x \mid x = x_k - t\, \nabla_x f(x_k),\ t \geq 0 \quad \text{and} \quad x \in \mathcal{B}_k\}. \tag{2.21}$$

If our model is a quadratic one of the form (2.17), then it is possible to calculate the exact minimum of the model $m_k$ along this arc. The computing minimizer $x_k^C$ is known as the *Cauchy point*. We will see in the following chapters that the Cauchy point is of essential importance in the convergence analysis of algorithms based on trust-region methods. Such methods are proved to be globally convergent if steps $s_k$ give a reduction in the model that is at least a fraction of the decrease obtained at the Cauchy point. More formally, the condition on the reduction in the model is given by

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa \left(m_k(x_k) - m_k(x_k^C)\right),$$

for some constant $\kappa \in (0, 1)$.

Note that, as stated, Algorithm 2.2 lacks a formal stopping criterion. In practice, one would obviously stop the process if the objective gradient's norm $\|\nabla_x f(x_k)\|$ falls below some tolerance, or if some fixed maximum number of iterations is exceeded. We will discuss in Chapter 5 reasonable values for the constants $\eta_1, \eta_2, \gamma_1$ and $\gamma_2$.

Finally, a complete description of practical methods to solve, exactly or approximately, the trust-region subproblem (2.18) is given in [34, Chapter 7].

### 2.3.4 Conjugate-gradient methods

To conclude this overview of methods for unconstrained optimization problems, we would like to present another popular class of algorithms, namely conjugate-gradient methods. The first motivation of the *conjugate-gradient method* consists in solving a linear system made up with a symmetric positive definite matrix and this method is due to Hestenes and Stiefel [81]. This process is performed without storing any additional matrix, even the matrix of the system. As a consequence one of the key properties of these methods is that they require little storage. Especially, when the number $n$ of variables is large, the conjugate-gradient method may surpass Newton methods.

The basis of conjugate-gradient algorithms is the notion of *conjugate directions*. A set of nonzero vectors $\{d_1, \ldots, d_k\}$ is said to be *conjugate* with respect to a symmetric positive definite matrix $A$, if

$$d_i^T A d_j = 0, \qquad \text{for all} \quad i \ \text{ and } \ j \ \text{ such that } \ i \neq j.$$

The conjugate directions can be used to solve a quadratic minimization problem of the form

$$\min_{x \in \mathbb{R}^n} q(x) = \frac{1}{2} x^T H x + c^T x, \tag{2.22}$$

where $H$ is supposed to be symmetric and positive definite. In accordance with the first-order optimality conditions for convex functions (see Theorem 2.4), the unique solution to the problem (2.22) is also the unique solution to the system of linear equations

$$Hx = -c.$$

The conjugate-gradient algorithm is obtained by choosing the successive direction vectors as a conjugate version of the successive gradients obtained as the method progresses. The directions are determined sequentially at each step of the iteration. At iteration $k$ one evaluates the current negative gradient vector and it is added a linear combination of the previous direction vectors to obtain a new conjugate direction vector along which to move.

The conjugate-gradient method is described in the following algorithm.

**Algorithm 2.3: Conjugate-gradient method**

Given a starting point $x_0$, set $g_0 = Hx_0 + c$ and $d_0 = -g_0$.
Until convergence, perform the following sequence of operations

$$\begin{aligned}
\alpha_k &= \frac{g_k^T g_k}{d_k^T H d_k} \\
x_{k+1} &= x_k + \alpha_k d_k \\
g_{k+1} &= g_k + \alpha_k H d_k \\
\beta_k &= \frac{g_{k+1}^T g_{k+1}}{g_k^T g_k} \\
d_{k+1} &= -g_{k+1} + \beta_k d_k \\
k &= k + 1
\end{aligned}$$

The convergence of this method must occur by the $n$th iteration in exact arithmetic.

## 2.4 Methods for nonlinear constrained optimization

We start by restating, for the sake of clarity, the structure of most constrained nonlinear optimization problems

$$\begin{aligned}
\min \quad & f(x) \\
\text{s.t.} \quad & c_i(x) = 0, \quad i \in \mathcal{E}, \\
& c_i(x) \leq 0, \quad i \in \mathcal{I},
\end{aligned} \tag{2.23}$$

in which both the objective and constraint functions may be nonlinear. Constrained optimization problems of that form arise naturally in many different disciplines. In this section, we give the principal ideas of some of well-known methods for nonlinear constrained optimization problems, in particular, of those that will be useful or mentioned in the remainder of this dissertation. Such methods are motivated by the desire to benefit from unconstrained optimization techniques to solve constrained problems; to this end, we can reformulate the problem (2.23) as an unconstrained one.

### 2.4.1 Penalty methods

We have chosen to present penalty methods because they will be considered in the section devoted to the motivation of filter techniques (see Section 3.1), since the latter have been introduced as an alternative to penalty-based approaches.
The main idea of penalty methods is to compute an approximate solution to a constrained optimization problem by successive unconstrained optimization problems defined by a combination of the original objective function and a function that penalizes the constraint violation. In other words, penalty methods try to solve simultaneously two issues, namely the problem of optimality and the one of feasibility. These techniques need to evaluate the "quality" of the current iterate with respect to previous ones. So in order to decide whether a candidate point $x_{k+1}$ is "better" than a current point $x_k$, it is necessary to define a so-called *penalty function* (or a *merit function*) which is designed to measure the balance between the often conflicting aims of decreasing the objective function and satisfying the constraints. To this end, we consider a function which assesses the constraint violation that we denote by $\theta(x)$ (see below for examples of this function). An advantage of penalty methods is that they do not require the iterates to be strictly feasible and they are thus appropriate for problems with equality constraints.

Penalty techniques then combine the two following problems

$$\min_x f(x) \qquad \text{and} \qquad \min_x \theta(x)$$

in a single one. This is realized by incorporating the measure of the constraint violation into the objective function. The function $\theta$ penalizes the constraint violation by imposing penalties for infeasibility : $\theta$ is zero on the feasible region and positive outside. The *penalty function* may then be defined as

$$f(x) + \rho\, \theta(x)$$

where the scalar $\rho$ is called the *penalty parameter*. A step will be accepted only if it results in a sufficient reduction of this penalty function. By varying the penalty parameter, the method converges iteratively to an approximate solution of the original constrained optimization problem.

So, more formally, we solve the following problem, where the value of the penalty parameter depends on the iteration,

$$\min_x f(x) + \rho_j \, \theta(x), \tag{2.24}$$

with the sequence $\{\rho_j\}$ tends to $+\infty$, in order to penalize the constraint violation more and more severely and to force the solution to be feasible. We hope that if the penalty parameter is sufficiently large, the solution of the problem (2.24) should be close to a solution of the original constrained problem (2.23). The determination of these parameters is usually a difficult task and its role is crucial for the performance of practical algorithms.

Many penalty functions exist, we shall only introduce the most common ones. We start with probably the most widely used, that is the *quadratic penalty function*

$$Q(x;\mu) \stackrel{\text{def}}{=} f(x) + \frac{1}{2\mu} \sum_{i \in \mathcal{E}} c_i^2(x) + \frac{1}{2\mu} \sum_{i \in \mathcal{I}} [\max(0, c_i(x))]^2, \tag{2.25}$$

where the penalty parameter is $\mu$. So, in quadratic penalty methods, we consider a sequence of parameters $\{\mu_k\}$ such that $\mu_k$ tends to zero as $k$ tends to $+\infty$. For each $k$, we find an approximate minimizer, denoted by $x_k$, of the problem

$$\min_x \; Q(x;\mu_k). \tag{2.26}$$

Note that, when only equality constraints are present, the quadratic penalty function (that is (2.25) without the last term) is smooth, but adding the inequality constraint term might result in a less smooth function[(1)]. A disadvantage of the method is that the Hessian of the penalty function becomes increasingly ill-conditioned close to the minimizer of $Q(x;\mu)$ when $\mu$ tends to zero. Both Quasi-Newton and conjugate-gradient methods are severely affected by this.

Another attractive merit function is the $\ell_1$ *exact penalty function* which is defined by

$$\ell_1(x;\mu) \stackrel{\text{def}}{=} f(x) + \frac{1}{\mu} \sum_{i \in \mathcal{E}} |c_i(x)| + \frac{1}{\mu} \sum_{i \in \mathcal{I}} \max(0, c_i(x)). \tag{2.27}$$

The use of the $\ell_1$ exact penalty function to solve problem (2.23) is well known. It has been first reported by Pietrzykowski [105], and has been widely studied, for example by Charalambous [21], Conn [27], Coleman and Conn [24, 25] and Han and Mangasarian [80]. This penalty function is called *exact* because, for some choices of the parameter $\mu$, $\ell_1(x;\mu)$ is minimized locally by the solution $x^*$ to the initial problem (2.23). This means that the exact solution of (2.23) may be obtained by one application of an unconstrained minimization procedure to the function $\ell_1(x;\mu)$ for $\mu$ sufficiently small. Note that the $\ell_1$ exact penalty function is nonsmooth.

[(1)] In fact, the Hessian of $Q(x;\mu)$ is piecewise twice continuously differentiable.

Finally, as we have already mentioned, one of the practical difficulties of penalty methods is the adjustment of the penalty parameter; several rules for updating this parameter during the process of the computation exist.

### 2.4.2 Augmented Lagrangian methods

In this section we introduce the main ideas of the augmented Lagrangian method as it is a major component of the solver LANCELOT which will be numerically compared with our filter-trust-region algorithms (see Sections 5.4.3 and 8.3.2).
For simplicity of presentation, we consider the problem with only equality constraints

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & c_i(x) = 0, \quad i \in \mathcal{E}. \end{array} \tag{2.28}$$

In methods with augmented Lagrangian function, we consider adding the penalty term to the Lagrangian function rather than to the objective function like, for example, in quadratic penalty methods. Therefore, the *augmented Lagrangian function* for the problem (2.28) is defined as

$$\mathcal{L}_A(x, \lambda; \mu) \stackrel{\text{def}}{=} f(x) + \sum_{i \in \mathcal{E}} \lambda_i c_i(x) + \frac{1}{2\mu} \sum_{i \in \mathcal{E}} c_i^2(x), \tag{2.29}$$

where $\mu > 0$ is the penalty parameter and $\lambda_i$ $(i \in \mathcal{E})$ are the Lagrange multipliers. Recalling the KKT conditions given in (2.5) (where we now only consider equality constraints), they require - among others - that

$$\nabla_x \mathcal{L}(x^*, \lambda^*) = 0 \ \text{ and } \ c_i(x^*) = 0 \ \ \forall i \in \mathcal{E},$$

where $\lambda^*$ denotes the optimal vector of Lagrange multipliers. Therefore the augmented Lagrangian and the Lagrangian coincide at the optimum, and we avoid the need to decrease $\mu$ to zero.

Computing the first derivative of the augmented Lagrangian with respect to $x$ at some non-critical point, we obtain

$$\nabla_x \mathcal{L}_A(x, \lambda; \mu) = \nabla_x f(x) + \sum_{i \in \mathcal{E}} \left[ \lambda_i + \frac{c_i(x)}{\mu} \right] \nabla_x c_i(x).$$

Consequently, close to an optimal solution, we expect that

$$\lambda_i^* \approx \lambda_i + \frac{c_i(x)}{\mu} \quad \text{for all } i \in \mathcal{E}.$$

Moreover, this formula suggests a technique to update the multipliers $\lambda_i$ in a sequence of iterates

$$\lambda_i^{k+1} = \lambda_i^k + \frac{c_i(x_k)}{\mu_k} \quad \text{for all } i \in \mathcal{E}, \tag{2.30}$$

where $\lambda_i^k$ denotes the $i$th component of the $k$th estimate.

This leads to the following framework for algorithms using augmented Lagrangian function.

**Algorithm 2.4: Augmented Lagrangian algorithm**

We are given $x_0^s$, $\lambda_0$ and $\mu_0 > 0$ and a convergence tolerance $\tau_0 > 0$. Set $k = 0$.

**Step 1 : Inner minimization.** Apply an unconstrained minimization algorithm to (approximately) solve the problem

$$\min_{x \in \mathbb{R}^n} \mathcal{L}_A(x, \lambda_k; \mu_k), \tag{2.31}$$

starting at $x_k^s$ and terminating at $x_k$ for which

$$\|\nabla_x \mathcal{L}_A(x_k, \lambda_k; \mu_k)\| \leq \tau_k.$$

If some final convergence test holds, STOP with (approximate) solution $x_k$.

**Step 2 : Update the Lagrange multiplier estimates and the penalty parameter.** Choose $\lambda_{k+1}$ according to (2.30). Choose a new penalty parameter such that $\mu_{k+1} \in (0, \mu_k]$. Select the next starting point as $x_{k+1}^s = x_k$. Increment $k$ by one and go to Step 1.

It has been proved that the convergence of the augmented Lagrangian method can be assured without decreasing the parameter $\mu$ to a very small value (see Nocedal and Wright [101]). Some motivations associated to these methods is that ill-conditioning is not unavoidable like in the quadratic penalty method, and the function we want to minimize is continuously differentiable in contrast with the $\ell_1$ exact penalty function given by (2.27). As said before, a well-known implementation of this method is that of Conn, Gould and Toint [32, 31] in the large-scale nonlinear programming package LANCELOT (Large And Nonlinear Constrained Extended Lagrangian Optimization Techniques) (see Section 5.4.3 for a brief description of this solver). It is also possible to extend the augmented Lagrangian method for inequality constraints. The problem (2.23) can be transformed into a problem including only equality constraints and bound constraints by using slack variables (see Nocedal and Wright [101]), and it can then be solved by LANCELOT.

### 2.4.3 Sequential Quadratic Programming

Since we will often refer to sequential quadratic programming in Chapter 3, we will present its main ideas in this section. Many of the most effective and popular methods for solving smooth nonconvex and nonlinear minimization problems with nonlinear constraints belong to the class of *Sequential Quadratic Programming (SQP)* techniques.

The SQP approach is a generalization of Newton's method to the constrained case and has been first introduced by Wilson in his Master's thesis [127]. Practically, it finds a trial step by minimizing a quadratic model of the problem. The quadratic program is a local model of the general constrained problem (2.23) at the iterate $x_k$, made up with a quadratic approximation of the objective function and a linear approximation of both equality and inequality constraints around the current iterate. The step $s_k$ is then computed by solving the following quadratic subproblem

$$\begin{array}{ll} \min\limits_{s\in\mathbb{R}^n} & \nabla_x f(x_k)^T s + \frac{1}{2} s^T H_k s \\ \text{s.t.} & c_{\mathcal{E}}(x_k) + A_{\mathcal{E}}(x_k) s = 0, \\ & c_{\mathcal{I}}(x_k) + A_{\mathcal{I}}(x_k) s \leq 0, \end{array} \tag{2.32}$$

where we gather into $c_{\mathcal{E}}$ and $c_{\mathcal{I}}$ the constraints functions $c_i (i \in \mathcal{E})$ and $c_i (i \in \mathcal{I})$ respectively and $A_{\mathcal{E}}(x_k)$ and $A_{\mathcal{I}}(x_k)$ are the Jacobian of the equality and inequality constraints, respectively, evaluated at $x_k$.

In the case of constrained optimization problems, one wants the next iterate not only to reduce the objective function $f$ but also to decrease some constraint violation. As these two goals are sometimes conflicting, it is required to assess the relative weight between them. Therefore, we use a so-called *merit function* defined in Section 2.4.1 as a criterion to determinate if the trial point is "better" than the current point. So when using SQP techniques, we need a merit function to evaluate the progress in improving the objective function while maintaining feasibility. For the sake of clarity, we focus our attention on equality-constrained problems (2.28). Two merit functions are commonly used in the framework of SQP algorithms, that is the augmented Lagrangian function (2.29) and the $\ell_1$ merit function (2.27), which for problems involving only equality constraints, is defined as

$$\ell_1(x;\mu) = f(x) + \frac{1}{\mu}\|c(x)\|_1.$$

One of these functions can be used to measure the progress towards a local minimum of the problem. An important question relates to the choice of the matrix $H_k$ in (2.32). One might choose $H_k$ to be the Hessian of the Lagrangian function, but it is also possible to set $H_k$ to an approximation of this Hessian.

For a complete survey of the topic, we refer the reader to the books of Nocedal and Wright [101, Chapter 18] and Conn, Gould and Toint [34, Chapter 15] or to the papers of Boggs and Tolle [12] and Gould and Toint [73]. Several software packages for solving large-scale constrained optimization problems use SQP methods, like, for example, SNOPT invented by Gill, Murray and Saunders [60].

### Globalization techniques

In order to ensure global convergence of SQP methods even if we start far from a solution, we must consider a globalization strategy as in the unconstrained case. The techniques described in Sections 2.3.2 and 2.3.3 may be adapted to SQP techniques; but here we only consider trust regions. The first trust-region SQP methods were introduced by Beale [5] and Sargent [112]. These methods solve the QP subproblem (2.32) with the additional constraint

$$\|s\| \leq \Delta_k.$$

The numerator of the ratio $\rho_k$ (2.19) must then be defined as the actual reduction in the merit function rather than in the objective function as for the unconstrained case. There are several ways to achieve this : one may modify the usual quadratic model (2.10) by adding a penalty term such that it is explicitly related to the merit function; another possibility is to transform the denominator of $\rho_k$ by including a term reflecting some predicted reduction in the constraint violation.

### Active-set SQP methods

One possible approach to solve the QP subproblem (2.32) at each iteration is to use active-set SQP methods. An active-set SQP algorithm solves a QP subproblem by using an inner process which repeatedly updates a set of constraint indices $\mathcal{W}$, known as the *working set*, which it estimates to be active at the solution of the QP subproblem. The iterations of this process are called the inner iterations and the working set at the $k$th inner iteration is denoted by $\mathcal{W}_k$.

For each inner iteration, we solve an *equality* constrained QP subproblem, in which the constraints corresponding to the current working set $\mathcal{W}_k$ are treated as equalities and all other constraints are temporarily ignored. These methods typically modify the working set by at most one constraint at each inner iteration. This is done either by adding a new constraint to the current working set $\mathcal{W}_k$ or by dropping a constraint from it. This process is repeated until the correct active set and the solution to (2.32) has been found. This limitation on the modification of the working set by at most one constraint at a time gives a lower bound on the number of inner iterations required to solve the QP subproblem. Indeed, if there are $k_0$ constraints active

at the initial working set $\mathcal{W}_0$ and $k^*$ constraints active at the solution, then at least $|k^* - k_0|$ iterations are needed for convergence. This feature can be an important disadvantage for large-scale problems if the initial working set is very different from the active set at the solution. However, some research works have led to methods allowing radical changes in the working set at each iteration (see Chapter 7).

For more details on active-set SQP methods, we refer the reader to the books of Nocedal and Wright [101]. Efficient implementations of active-set SQP methods such as FilterSQP [52, 50] and SNOPT [60] have been developed.

## 2.5 References

Obviously, the survey of this chapter is not thorough and, for a more complete study of methods and algorithms devoted to nonlinear optimization, we refer the reader to the books of Bertsekas [8], Bonnans, Gilbert, Lemaréchal and Sagastizábal [14], Conn, Gould and Toint [34], Dennis and Schnabel [41], Fletcher [48], Gill, Murray and Wright [62], Luenberg [91], Nash and Sofer [98], Nocedal and Wright [101] and Ruszczyński [108] and to the papers of Gould et al. [71] and Sartenaer [114].

To conclude this chapter, we give a list of online resources and tools related to nonlinear optimization. We first mention the NEOS (Network-Enabled Optimization System) Guide and the NEOS Server that have been developed by the Optimization Technology Center[(2)]. They are available at the following addresses :

`http://www.mcs.anl.gov/otc/Guide/`

and

`http://www-neos.mcs.anl.gov/`

The former contains information and educational material about optimization, including a guide to optimization software. The NEOS Server permits to solve optimization problems remotely over the Internet. The user can submit a problem, for example, through the World Wide Web. A few moments later, he receives the result obtained by the chosen optimization solver. Another interesting link is the following :

`http://www.optimization-online.org/`

Optimization Online is a repository of e-prints about optimization and related topics. It allows researchers to announce their new reports.

[(2)] `http://www.ece.northwestern.edu/OTC`

# Chapter 3

# A quick survey of filter methods

This chapter is intended to provide an introduction to the concept of *filter*, which is the discussion thread of this work. The algorithms we propose in Chapters 4 and 7 are principally based on this notion. We will start with a motivation of the introduction of the filter in optimization. Afterwards, we will give the main ideas of the first filter-based algorithm proposed by Fletcher and Leyffer [52]. We will then conclude this chapter with a review of the existing methods and algorithms using a filter, which is presented in Section 3.3.

## 3.1 Motivation of the filter

The use of a penalty function in order to reach global convergence is a common feature of most algorithms for constrained nonlinear optimization. However, as we have seen in the previous chapter, one of the major difficulties of methods using a penalty function lies in the choice of the sequence of penalty parameters $\{\rho_k\}$ and, in particular, in the choice of a suitable initial parameter $\rho_0$. Another consequence of the application of penalty methods is the effect of the nondifferentiability of some exact penalty functions such as those encountered in Section 2.4.1. On the other hand, Zoppke-Donaldson presented in his PhD thesis [132] a practical SQP algorithm that does not require the use of a penalty function. This indicates that a pure SQP method can work very well and quickly in practice.

In order to get around these issues due to penalty functions, Fletcher and Leyffer [52] have introduced, in the last nineties, a new method based on the concept of a *filter*. The latter allows to avoid the use of a merit function to guarantee global convergence in algorithms for nonlinear programming. As we have already pointed out, there are two potentially conflicting aims in nonlinear constrained programming : the minimization of the objective function and the satisfaction of the constraints. Filter techniques consider these two goals separately and thus allow

contradictory objectives. In a way, the filter can be seen as a tool inspired from multicriteria optimization (see, *e.g.* Miettinen [93]). Indeed, filter techniques borrow the concept of *dominance* from this field to create the filter.

The approach of Fletcher and Leyffer aims to interfere as few as possible with the underlying Newton-type method. So one of the major motivations for the development of filter methods is to make them more permissive than approaches based on a merit function. This is why, in filter methods for constrained optimization, a new point is accepted if either the objective function is reduced and/or the constraints violation is improved compared to all previous iterates, contrary to penalty approaches, where a trial point is accepted if it reduces a function defined by a weighted combination of these two measures. The most important point in filter methods is to give up the strict monotone behaviour of usual measures, like penalty functions. The filter then allows to increase the flexibility in optimization processes to accept new iterates and generally allows larger steps towards the solution.

More formally, we can say that the notion of filter is based on that of *dominance*. To illustrate this fact, imagine a situation where one would like to reduce at the same time two potentially conflicting objectives, denoted by $\theta_1(x)$ and $\theta_2(x)$. We say that a point $x$ *dominates* a point $y$ if and only if

$$\theta_i(x) \leq \theta_i(y) \qquad \text{for } \; i = 1, 2.$$

Thus, if we focus our attention on reducing both $\theta_1$ and $\theta_2$ and if point $x$ dominates point $y$, the latter is of no real interest to us since $x$ is at least as good as $y$ with respect to both objectives. So all we need to do is to remember iterates that are not dominated by other iterates by using the filter. Algorithms based on this notion of filter have to store the pairs $(\theta_1, \theta_2)$ corresponding to successful previous iterations. Figure 3.1 illustrates the notion of dominance, where we have simplified the notation by using $x$ to represent the pair $(\theta_1(x), \theta_2(x))$. The point y is dominated by $x$ because

$$\theta_1(y) > \theta_1(x) \quad \text{and} \quad \theta_2(y) > \theta_2(x),$$

while the point $z$ is not dominated by $x$ since

$$\theta_1(z) > \theta_1(x) \quad \text{but} \quad \theta_2(z) < \theta_2(x).$$

## 3.2 The first filter approach

The first filter approach was proposed by Fletcher and Leyffer [52] and was devoted to the solution of nonlinear constrained problems by a sequential quadratic programming (SQP) trust-

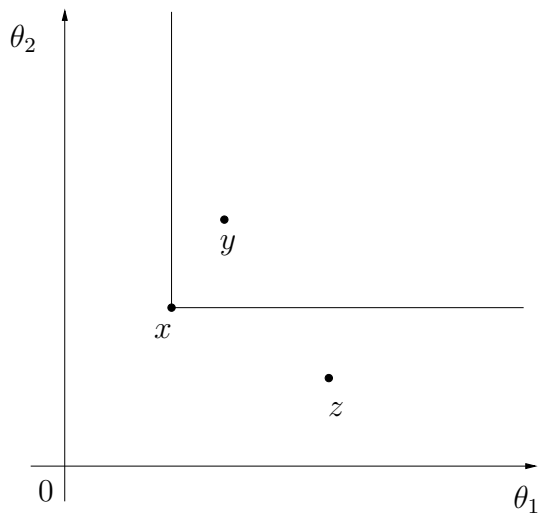


Figure 3.1: The notion of dominance

region algorithm (see Section 2.4.3). They wanted to replace the use of a merit function in the SQP framework, whose goal is to guarantee a balance between decreasing the objective function and reducing the infeasibility. This work provides the essential ideas that are the foundations of all other filter algorithms developed in the last decade.

For simplicity, we restrict our attention to inequality-constrained problems of the form

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & c_i(x) \leq 0, \quad i \in \mathcal{I}, \end{array} \tag{3.1}$$

where the objective and the constraints are smooth. In this case, the two objectives $\theta_1$ and $\theta_2$ become, respectively,

$$f(x) \qquad \text{and} \qquad \sum_{i \in \mathcal{I}} \max(0, c_i(x)).$$

Obviously, other choices are possible for measuring the constraint violation $\theta_2$ (see, for instance, Gould and Toint [74]). So, as we have already said, the filter approach views $\theta_1$ and $\theta_2$ as individual purposes with a slight priority to the minimization of $\theta_2$ because we must find a point $x^*$ satisfying $\theta_2(x^*) = 0$. In the remaining part of this section, $\theta_1$ will be denoted by $f$ and $\theta_2$ by $\theta$.

So, for inequality-constrained optimization, we say that a point $x_k$ *dominates* a point $x_l$ whenever

$$\theta(x_k) \leq \theta(x_l) \qquad \text{and} \qquad f(x_k) \leq f(x_l).$$

Hence the *filter* is defined as a list of pairs $(\theta(x_i), f(x_i))$ such that no pair dominates another one. We denote by $(\theta_i, f_i)$ the $(\theta, f)$-pair associated to $x_i$. More formally, the filter is character-

ized by

$$\mathcal{F} = \{(\theta_i, f_i) \quad \text{such that} \quad \theta_i < \theta_j \quad \text{or} \quad f_i < f_j \quad \text{for} \quad i \neq j\}\,. \tag{3.2}$$

We thus accept a new iterate only if it is not dominated by any other iterate already in the filter.

The filter determines a *forbidden region* in the $(\theta, f)$-space by memorizing some pairs from the previous iterates. On plots of Figure 3.2, the shaded areas illustrate the region where points have to be rejected.

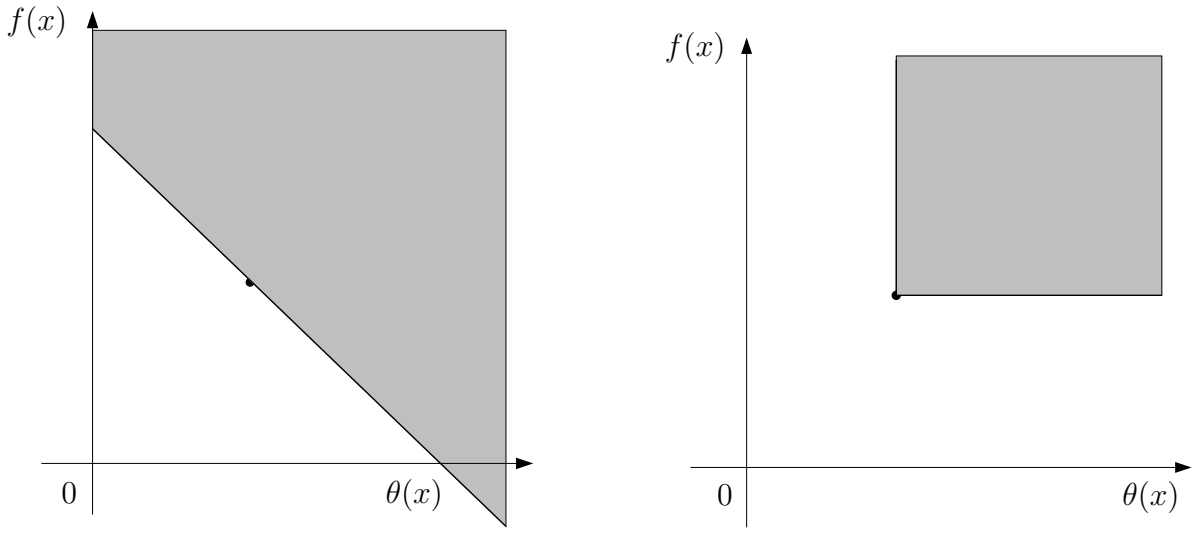


Figure 3.2: Forbidden regions defined by a penalty function and a filter

The plot on the left represents the isovalue of an exact penalty function of the form

$$f(x) + \rho\, \theta(x)$$

through the current point in the $(\theta, f)$-space, that is a straight line with a slope equal to the opposite of the penalty parameter $\rho$. Points belonging to the region to the left of this line reduce the penalty function value. The plot on the right shows the area dominated by the filter entry. We can clearly observe that, for an entry in the $(\theta, f)$-space, the forbidden region defined by a penalty function is larger than that determined by the filter, so methods using a penalty function are generally more restrictive than filter approaches.

The basic idea of Fletcher and Leyffer is that the filter plays the role of a criterion for accepting or rejecting a trial step in an SQP method (see Section 2.4.3). At an iteration $k$, SQP

approaches solve a quadratic approximation of (3.1) within a trust region :

$$\begin{array}{ll} \min\limits_{s\in\mathbb{R}^n} & \nabla_x f(x_k)^T s + \frac{1}{2} s^T H_k s \\ \text{s.t.} & c_{\mathcal{I}}(x_k) + A_{\mathcal{I}}(x_k) s \leq 0 \\ & \|s\|_\infty \leq \Delta_k, \end{array} \tag{3.3}$$

where $H_k$ is some approximation of the Hessian of the Lagrangian function and $A_{\mathcal{I}}(x_k)$ is the Jacobian of the constraints $c_{\mathcal{I}}$ evaluated at $x_k$. The solution of this trust-region subproblem gives a trial step $s_k$ towards the new iterate $x_{k+1}$. This trial point is accepted if it is not dominated by the current filter; the trust region is possibly increased and the filter is updated in the sense that the previous iterate is added to the filter and all entries dominated by this new one are removed from it. Otherwise, if the trial point is not acceptable for the filter, the step is rejected, the trust-region radius is decreased and we solve the new trust-region subproblem. In that way, the common rule on the reduction of a penalty function is replaced by the restriction that the point be acceptable for the filter.

However, many features of this simple framework have to be specified and refined so as to produce an efficient and globally convergent algorithm. In fact, we do not wish to accept a new point if it is arbitrarily close to being dominated by another point already in the filter. In fact, the definition of the filter (3.2) may permit points to accumulate in the vicinity of a filter entry where $\theta_k > 0$ and hence convergence to infeasible limit points. In order to avoid this situation, a *margin* may be adjusted along the filter. With this new concept, we then say that a trial point $x$ is *acceptable for the filter* $\mathcal{F}$ if and only if

$$\theta(x) < \beta\, \theta_j \;\text{ or }\; f(x) < f_j - \gamma\, \theta(x) \;\text{ for all }\; (\theta_j, f_j) \in \mathcal{F},$$

where $\beta$ and $\gamma$ are parameters in $(0, 1)$ with $\beta$ close to one and $\gamma$ close to zero. This definition of the margin comes from Chin and Fletcher [22]. The filter may be represented in the $(\theta, f)$-space as illustrated in Figure 3.3. The pairs $(\theta(x_k), f(x_k))$ are represented by black dots and the margin[(1)] is shown by the dashed line.

In this first work on filter methods, some other refinements have been made. Fletcher and Leyffer have also considered an upper bound on the constraint violation in order to avoid situations where a sequence of iterates satisfying

$$f(x_{k+1}) < f(x_k) \quad \text{and} \quad \theta(x_{k+1}) > \theta(x_k)$$

with $\theta(x_k)$ tending to infinity is accepted. On the other hand, considering only the filter acceptance mechanism would allow convergence to a feasible but non-optimal point. Indeed, this

[(1)] For the sake of clarity, we have increased the margin on the figure.

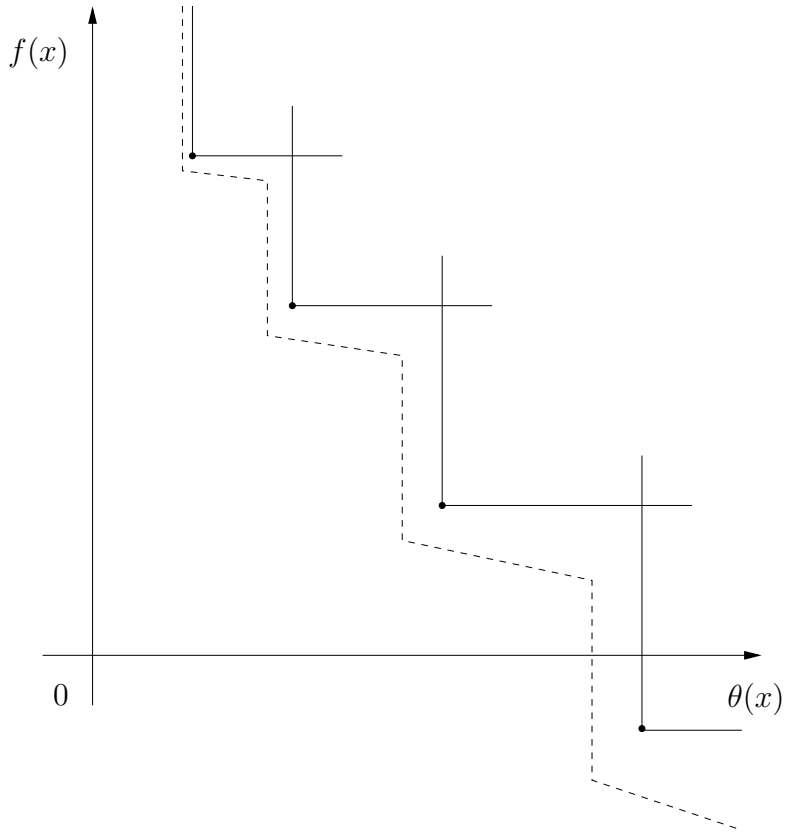


Figure 3.3: A filter with four pairs $(\theta, f)$

may happen if we accept a sequence of iterates $\{x_k\}$ such that they yield sufficient reduction in the constraint violation, *i.e.* such that $\theta(x_{k+1}) \leq \beta\theta(x_k)$. In order to avoid this situation, a *sufficient reduction* condition should be imposed whenever the constraint violation becomes small (see, for example, Wächter and Biegler [126] and Gould and Toint [74]). Another unsuitable effect of this technique is that the quadratic trust-region subproblem (3.3) may become inconsistent when reducing the trust-region radius. The strategy, always proposed by Fletcher and Leyffer, is to enter a *feasibility restoration phase* in which we aim to get closer to the feasible region by minimizing the constraint violation $\theta(x)$. This restoration phase thus attempts to find a new point $x_{k+1}$, acceptable for the filter, at which the quadratic trust-region subproblem is compatible for some radius $\Delta_{k+1} > 0$. An in-depth description of the algorithm and a discussion of these refinements may be found in the paper of Fletcher and Leyffer [52]. The implementation of this filter method, named filterSQP, and some numerical experiments are reported in Fletcher and Leyffer [50].

In order to obtain global convergence properties of filter methods, we shall generally require

that the following assumptions are satisfied :

- the functions $f$ and $c$ are twice continuously differentiable on $\mathbb{R}^n$;
- the iterates $x_k$ remain in a closed, bounded domain of $\mathbb{R}^n$;
- for all $k$, the model of the objective function $m_k$ is twice differentiable on $\mathbb{R}^n$ and has a uniformly bounded Hessian.

The fast local convergence of filter methods has been discussed in Ulbrich [123] and in Wächter and Biegler [125]. In this latter paper, the authors use second-order correction steps to achieve fast local convergence.

## 3.3 Bibliographical review

Since filter methods have been introduced for nonlinear constrained optimization by Fletcher and Leyffer [52], they have enjoyed considerable interest in their original domain of application as well as in other areas of optimization. A global convergence theory for a filter-based algorithm was first proposed in Fletcher et al. [54] : in this approach, the objective function was locally approximated by a linear function. The authors combined the use of the filter with a sequential linear programming (SLP) method. Chin and Fletcher [22] have analysed an SLP-filter algorithm that takes equality constrained quadratic programming (EQP) steps. In [55], Fletcher et al. have extended these methods to trust-region sequential quadratic programming (SQP) techniques and global convergence has been proved supposing that the quadratic trust-region subproblems are solved globally. In another paper of Fletcher et al. [49], the approach allows for an approximate solution of the quadratic subproblems and the global convergence to first-order critical points has also been proved. In this method, a Byrd-Omojokun-like approach (see Omojokun [103]) is followed. This means that the step is viewed as the sum of two components, a *normal* step $n_k$, such that $x_k + n_k$ satisfies the linearized constraints in the quadratic subproblem, and a *tangential* step $t_k$, whose goal is to decrease the value of the objective function's model while continuing to satisfy the linearized constraints. More formally, the step is computed as $s_k = n_k + t_k$. A second algorithm is also proposed in this paper, which replaces the decomposition into normal and tangential steps by a stronger condition on the associated model decrease. Another filter method combined with an SQP algorithm was also proposed by Ulbrich [123].

The generality of the filter concept also permits its adaptation to line-search techniques as well as to interior-point and active-set methods. Wächter and Biegler [126, 125] have presented and analysed a scheme for line-search filter methods that can be applied to barrier interior-point and active-set SQP algorithms. The implementation of this line-search filter method and some numerical experiments are discussed in Wächter's PhD thesis [124]. In [64], Gonzaga, Karas and Vanti have proposed a filter algorithm where each iteration is composed with a restoration phase, whose aim is to reduce some measure of infeasibility, and an optimal phase, which reduces the objective function in a tangential approximation of the feasible set. More recently, Ribeiro et al. [107] have extended the work of Gonzaga et al. [64]. Their general filter algorithm allows a great deal of freedom in the computation of the step. As we have already mentioned, filter methods have also been applied to interior-point frameworks by Wächter and Biegler [126, 125], Ulbrich et al. [122] and also by Benson et al. [6]. A non-monotone variant of the trust-region SQP-filter algorithm designed in [49] is proposed in Gould and Toint [75]. In this paper, the filter acceptance criterion for new iterates is relaxed in such a way that some points that would be rejected are now accepted. On the other hand, it is not necessary anymore to define a margin around the filter, rather the authors define, for a new entry of the filter, an area that represents its contribution to the dominated region.

Filter-type techniques have also been applied in other fields of optimization. They have been used, for example, by Fletcher and Leyffer [53] to solve a system of algebraic equations and inequalities. Gould et al. [65] have developed a multidimensional filter algorithm for the nonlinear feasibility problem (including systems of nonlinear equations and nonlinear least-squares), which is to minimize the norm of the violations of a set of (possibly nonlinear and/or nonconvex) constraints. Numerical results are also presented in [76] and indicate substantial gains in efficiency over the traditional monotone trust-region algorithm. Fletcher and Leyffer [51] have proposed a bundle filter method for nonsmooth optimization. Finally, in the framework of derivative-free optimization, Audet and Dennis [3] have used the filter idea combined with their pattern search algorithms (see [2]) and Colson [26] has proposed a filter-SQP algorithm in his PhD thesis in 2003.

Since the filter is a recent tool, the research in this field is highly active and many works are carried out on this topic every year. In the following chapters, filter methods based on trust-region frameworks are proposed and analysed.

# Part I

# Unconstrained Optimization

# Chapter 4

# A filter-trust-region method for unconstrained optimization

As emphasized in Chapter 3, filter methods have proved to be robust and efficient in nonlinear optimization. The focus of this chapter is the description and analysis of the filter-trust-region algorithm we designed for solving unconstrained nonlinear optimization problems. This approach is based on the filter technique introduced by Fletcher and Leyffer [52] (see Section 3.2). As we have mentioned in the previous chapter, Gould, Leyffer and Toint [65, 76] have proposed a filter method for the general smooth nonlinear feasibility problem. Formally, this problem is finding a vector $x \in \mathbb{R}^n$ such that

$$c_{\mathcal{E}}(x) = 0 \qquad \text{and} \qquad c_{\mathcal{I}}(x) \geq 0, \tag{4.1}$$

where $c_{\mathcal{E}}(x)$ and $c_{\mathcal{I}}(x)$ are smooth functions from $\mathbb{R}^n$ into $\mathbb{R}^m$ and $\mathbb{R}^q$, respectively. If this is not possible to find such a point, we can find a local minimizer of the constraint violation. The authors propose to find a local minimizer of the following problem

$$\min_{x \in \mathbb{R}^n} \ \frac{1}{2} \|\theta(x)\|_2^2, \tag{4.2}$$

where they define

$$\theta(x) \stackrel{\text{def}}{=} \begin{pmatrix} c_{\mathcal{E}}(x) \\ \min(0, c_{\mathcal{I}}(x)) \end{pmatrix} \in \mathbb{R}^{m+q}.$$

Note that when $q = 0$, the problem (4.1) reduces to a system of smooth nonlinear equations.

An efficient method for solving (4.1) or (4.2) is to use Newton's method because of its fast convergence. However, such a method has to be safeguarded in order to obtain global convergence. As is well-known, many safeguarding techniques are possible, like line searches

or trust regions (see Section 2.3.2 and 2.3.3). In [65], Gould et al. propose to combine the basic trust-region algorithm with the filter idea for solving problems (4.1) or (4.2). This algorithm can be found in the FILTRANE package [76], which is available as part of the GALAHAD library at

`http://galahad.rl.ac.uk/`

It is the purpose of this chapter to consider the further extension of filter techniques to general unconstrained optimization problems. The filter-trust-region algorithm described in this chapter was first introduced in the paper of Gould, Sainvitu and Toint [72].

We will start this chapter by describing the problem and its resolution by our new algorithm proposal and we will also adapt the definition of the filter to our context. Section 4.2 will address the theoretical aspects regarding the convergence of the proposed method.

## 4.1 The problem and the new algorithm

This chapter is intended to describe a novel algorithm for solving unconstrained minimization problems, that is mathematical programs of the following form

$$\min_{x \in \mathbb{R}^n} \; f(x), \tag{4.3}$$

where $f$ is a twice continuously differentiable function of the variables $x \in \mathbb{R}^n$. The leading ingredients of the proposed algorithm are :

- a *trust-region* framework for unconstrained optimization problems as described in Section 2.3.3;
- a *filter* technique (as defined in Chapter 3).

This last component has proved to be important because it allows the design of new non-monotone methods for optimization.

As we have already indicated in Section 2.3, an efficient technique for solving the unconstrained mathematical program (4.3) is to use Newton's method. Unfortunately, it is well-known that such an algorithm may not always be well-defined, when the Taylor's model is nonconvex, or convergent from any initial point $x_0$. As mentioned in Section 2.3.3, these difficulties can be circumvented by using a trust region, in a manner that is now well established (see Conn, Gould and Toint [34] for an extensive description of trust-region methods and their properties).

If we consider convex mathematical programs, we know from Theorem 2.4 that finding the unique minimizer of $f$ is equivalent to finding a zero of the gradient of the objective function. In other words, in convex programming, solving problem (4.3) is equivalent to finding a point $x^*$ such that

$$\nabla_x f(x^*) = 0.$$

This problem can be seen as the potentially conflicting aims of zeroing each component of the gradient $[\nabla_x f(x)]_i \ (i = 1, \ldots, n)$. So we propose a new unconstrained algorithm by introducing a *multidimensional filter* technique inspired by that presented in [65]. The aim of the filter here is to encourage convergence of iterates to first-order critical points by driving every component of the objective's gradient

$$\nabla_x f(x) \stackrel{\text{def}}{=} g(x) = (g_1(x), \ldots, g_n(x))^T$$

to zero. So each entry of our multidimensional filter is a component of the gradient, which corresponds to the following choice in the definition of the filter

$$\theta_i(x) = |(\nabla_x f(x))_i| \quad i = 1, \ldots, n.$$

### 4.1.1 Computing a trial point

The computation of the trial point

$$x_k^+ = x_k + s_k$$

is performed by using trust-region techniques described in Section 2.3.3. At variance with classical trust-region methods, we do not require here that

$$\|s_k\| \leq \Delta_k \tag{4.4}$$

at every iteration of our algorithm. In this definition and below, $\|\cdot\|$ stands for the Euclidean norm. The convergence analysis given in Section 4.2 requires, as is common in trust-region methods (see [34, Chapter 6]), that the step $s_k$ provides, at iteration $k$, a *sufficient decrease on the model*, which is to say that

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{\text{mdc}} \max \left[ \|g_k\| \min \left[ \frac{\|g_k\|}{\beta_k}, \Delta_k \right], |\tau_k| \min[\tau_k^2, \Delta_k^2] \right] \tag{4.5}$$

where $\kappa_{\text{mdc}}$ is a constant in $(0, 1)$, $\beta_k$ is a positive upper bound on the norm of the Hessian of the model $m_k$, *i.e.*

$$\beta_k \stackrel{\text{def}}{=} 1 + \|H_k\|, \tag{4.6}$$

and

$$\tau_k \stackrel{\text{def}}{=} \min\left[0, \lambda_{\min}(H_k)\right], \tag{4.7}$$

where $\lambda_{\min}(\cdot)$ denotes the smallest eigenvalue.
Although this condition on the model seems technical, there are efficient numerical methods to compute $s_k$ that guarantee that (4.5) holds (see Section 2.3.3, Gould et al. [66], Moré and Sorensen [95], or, more generally, [34, Chapter 7]).

### 4.1.2 The multidimensional filter

We now describe the way we define and use the filter idea in the context of unconstrained nonlinear optimization. We consider using a filter mechanism to potentially accept $x_k^+$ as the new iterate more often than in a classical trust-region method. As we have seen in Chapter 3, the notion of filter is based on that of *dominance*: for the problem under study in the present chapter, we say that a point $x_1$ *dominates* a point $x_2$ whenever

$$|g_i(x_1)| \leq |g_i(x_2)| \quad \text{for all} \quad i = 1, \ldots, n.$$

Thus, if iterate $x_1$ dominates iterate $x_2$ and if we focus our attention on convergence to first-order critical points only, the latter is of no real interest to us since $x_1$ is at least as good as $x_2$ for each of the components of the gradient. All we need to do is to remember iterates that are not dominated by other iterates by using the dynamic structure of the filter. So, in this context of unconstrained optimization, we define a *multidimensional filter* $\mathcal{F}$ as a list of $n$-tuples of the form $(g_{k,1}, \ldots, g_{k,n})$, where $g_{k,i} \stackrel{\text{def}}{=} g_i(x_k)$, such that, if $g_k$ and $g_\ell$ belong to $\mathcal{F}$, then

$$|g_{k,j}| < |g_{\ell,j}| \quad \text{for at least one} \quad j \in \{1, \ldots, n\}. \tag{4.8}$$

Our filter method proposes to accept a trial iterate $x_k^+$ if it is not dominated by any other iterate in the filter.

In the previous chapter, we have pointed out the fact that, from an algorithmic point of view, we do not wish to accept a new point $x_k^+$ if one of the components of $g(x_k^+)$ is arbitrarily close to being dominated by another point already in the filter. So again, in order to avoid this situation, we slightly strengthen our acceptability test and we say that a new trial point $x_k^+$ is *acceptable for the filter* $\mathcal{F}$ if and only if

$$\forall g_\ell \in \mathcal{F} \qquad \exists\, j \in \{1, \ldots, n\} \quad \text{such that} \quad |g_j(x_k^+)| < |g_{\ell,j}| - \gamma_g \|g_\ell\|, \tag{4.9}$$

where $\gamma_g \in (0, 1/\sqrt{n})$ is a small positive constant. The margin is measured as a function of the objective's gradient at the existing filter point. Because of the upper limit $1/\sqrt{n}$ imposed

on $\gamma_g$, one can be sure that the right-hand side of (4.9) is always positive for some index $j$ and therefore that points acceptable for the filter always exist.
In order to avoid cycling, and supposing the current point $x_k$ is acceptable for the filter in the sense of (4.9), we may wish to add it to the filter so as to exclude other worse iterates; this is simply done by the updating formula :

$$\mathcal{F} := \mathcal{F} \cup \{g(x_k)\}.$$

We remove from the filter every $g_\ell \in \mathcal{F}$ such that

$$|g_{j,\ell}| \geq |g_{j,k}| \quad \text{for all} \quad j \in \{1, \ldots, n\}.$$

A filter and its margins[(1)], which increase with $\|g\|$, are represented in Figure 4.1 for a two-dimensional setup, *i.e.* $x \in \mathbb{R}^2$.

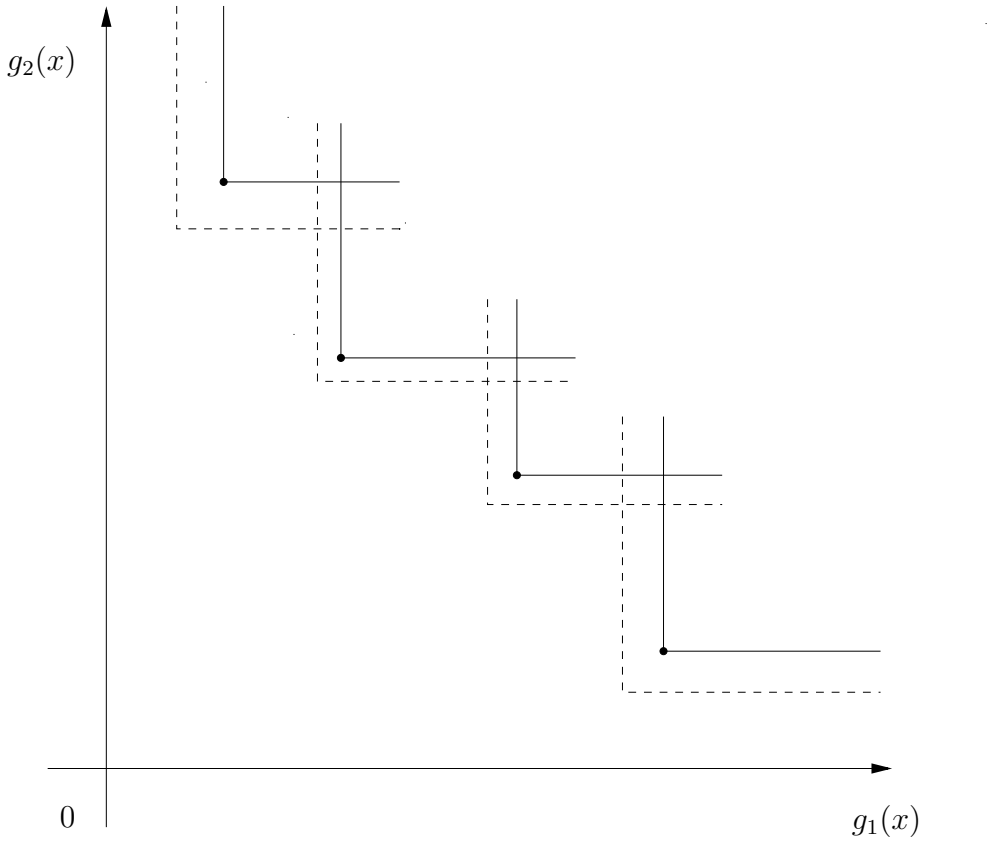


Figure 4.1: A filter for an unconstrained problem in $\mathbb{R}^2$

[(1)]For the sake of clarity, we have increased the margins.

However, the process described so far only guides the iterates towards a zero of the gradient $\nabla_x f(x)$. This is adequate for convex problems, where a zero gradient is both necessary and sufficient for second-order criticality (see Theorem 2.4), but it may be unsuitable for nonconvex ones. Indeed it might prevent progress away from a saddle point, in which case an increase in the gradient components is acceptable. We therefore modify the filter mechanism presented above to ensure that the filter is reset to the empty set after each iteration giving sufficient decrease in the objective function at which the model $m_k$ was detected to be nonconvex. We also set an upper bound on the acceptable objective function values to ensure that the achieved decrease is permanent (see Algorithm 4.1).

### 4.1.3 The filter-trust-region algorithm

Now that we have presented the principal ingredients of our algorithm, we may summarize it as follows : the main objective of our algorithm for unconstrained optimization is to let the filter play the major role in ensuring global convergence within "convex basins" (*i.e.* when convexity is present), and fall back on a usual trust-region method only if things do not go well or if negative curvature is encountered during the minimization of the trust-region subproblem.

A more detailed setup of our algorithm follows :

**Algorithm 4.1: Filter-Trust-Region Algorithm**

**Step 0: Initialization.** An initial point $x_0$ and an initial trust-region radius $\Delta_0 > 0$ are given. The constants $\gamma_g \in (0, 1/\sqrt{n})$, $\eta_1, \eta_2, \gamma_1, \gamma_2$ and $\gamma_3$ are also given and satisfy

$$0 < \eta_1 \leq \eta_2 < 1 \quad \text{and} \quad 0 < \gamma_1 \leq \gamma_2 < 1 \leq \gamma_3.$$

Compute $f(x_0)$ and $g(x_0)$, set $k = 0$. Initialize the filter $\mathcal{F}$ to the empty set and choose $f_{\sup} \geq f(x_0)$. Define two flags `RESTRICT` and `NONCONVEX`, the former to be unset.

**Step 1: Determine a trial step.** Compute a finite step $s_k$ that "sufficiently reduces" the model $m_k$, *i.e.* that satisfies (4.5) and that also satisfies $\|s_k\| \leq \Delta_k$ if `RESTRICT` is set or if $m_k$ is nonconvex. In the latter case, set `NONCONVEX`; otherwise unset it. Compute the trial point $x_k^+ = x_k + s_k$.

**Step 2:** Compute $f(x_k^+)$ and define the following ratio

$$\rho_k = \frac{f(x_k) - f(x_k^+)}{m_k(x_k) - m_k(x_k^+)}.$$

If $f(x_k^+) \geq f_{\sup}$, set $x_{k+1} = x_k$, set `RESTRICT` and go to Step 4.

**Step 3: Test to accept the trial step.**

- Compute $g_k^+ = g(x_k^+)$.
- If $x_k^+$ is acceptable for the filter $\mathcal{F}$ and `NONCONVEX` is unset:
  Set $x_{k+1} = x_k^+$, unset `RESTRICT` and add $g_k^+$ to the filter $\mathcal{F}$ if either $\rho_k < \eta_1$ or $\|s_k\| > \Delta_k$.
- If $x_k^+$ is not acceptable for the filter $\mathcal{F}$ or `NONCONVEX` is set:
  If $\rho_k \geq \eta_1$ and $\|s_k\| \leq \Delta_k$, then

  set $x_{k+1} = x_k^+$, unset `RESTRICT` and if `NONCONVEX` is set, set $f_{\sup} = f(x_{k+1})$ and reinitialize the filter $\mathcal{F}$ to the empty set;

  else set $x_{k+1} = x_k$ and set `RESTRICT`.

**Step 4: Update the trust-region radius.** If $\|s_k\| \leq \Delta_k$, update the trust-region radius by choosing

$$\Delta_{k+1} \in \begin{cases} [\gamma_1\Delta_k, \gamma_2\Delta_k] & \text{if} \quad \rho_k < \eta_1, \\ [\gamma_2\Delta_k, \Delta_k] & \text{if} \quad \rho_k \in [\eta_1, \eta_2), \\ [\Delta_k, \gamma_3\Delta_k] & \text{if} \quad \rho_k \geq \eta_2; \end{cases} \tag{4.10}$$

otherwise, set $\Delta_{k+1} = \Delta_k$. Increment $k$ by one and go to Step 1.

Note that, as stated, our algorithm lacks a formal stopping criterion. In practice, one would obviously stop the calculation if $\|g_k\|$ falls below some user-defined tolerance (see Section 5.3) and the flag `NONCONVEX` is unset, or if some fixed maximum number of iterations is exceeded. Also note that our condition on the step might impose to recompute $s_k$ within the trust region if negative curvature was discovered for the model only after computing a step beyond the trust-region boundary. Fortunately, this is typically a very cheap calculation and it can be achieved by backtracking (see Nocedal and Yuan [102]) or by other suitable restriction techniques (see Gould et al. [66]). Details about the strategy used will be given in Section 5.3.

Remark that the filter collects information on selected previous iterates. Assuming the trial

point $x_k^+$ is acceptable for the filter and the flag NONCONVEX is unset, we might include this point in the filter in order to avoid, in the future, other iterates that are worse. Remembering that one of the motivations of the filter is to make filter-based approaches more permissive than penalty methods, we do not want to insert a point in the filter at each iteration. Therefore, we augment the filter for all points acceptable for the filter and for which negative curvature has not been detected during the computation of the trial step either if the reduction in the objective function is not sufficient or if the trial step is outside the trust region.

However, this algorithm only yields a framework for a practical implementation. The software resulting from the Fortran 90 implementation of this algorithm is named FILTRUNC and its practical aspects are discussed in Section 5.3.

## 4.2 Convergence analysis

This section is devoted to the theoretical aspects of Algorithm 4.1. Its global convergence properties will be proved under mild assumptions. Many of the proofs come from or are inspired by the convergence analysis of the basic trust-region algorithm (see [34, Chapter 6]).

### 4.2.1 Assumptions and notations

Let us first state the general assumptions that are necessary for the global convergence analysis of our filter-trust-region algorithm for unconstrained optimization.

**A1** The objective function $f$ is twice continuously differentiable on $\mathbb{R}^n$.

**A2** The iterates $x_k$ remain in a closed, bounded domain of $\mathbb{R}^n$.

**A3** For all $k$, the model $m_k(x)$ is twice differentiable on $\mathbb{R}^n$ and has a uniformly bounded Hessian.

Note that A1, A2 and A3 together imply that there exist constants $\kappa_{\text{l}}$, $\kappa_{\text{u}} \geq \kappa_{\text{l}}$, $\kappa_{\text{ufh}} \geq 1$ and $\kappa_{\text{umh}} \geq 1$ such that

$$f(x) \in [\kappa_{\text{l}}, \kappa_{\text{u}}], \quad \|\nabla^2_{xx} f(x)\| \leq \kappa_{\text{ufh}} \quad \text{and} \quad \|\nabla^2_{xx} m_k(x)\| \leq \kappa_{\text{umh}} - 1 \tag{4.11}$$

for all $k$ and all $x$ in the convex hull of $\{x_k\}$. Combining this with the definition of $\beta_k$ given in (4.6), we have that

$$\beta_k \leq \kappa_{\text{umh}} \tag{4.12}$$

for all $k$.
For the sake of simplicity, the sufficient decrease on the model $m_k$ given in (4.5) is restated in the following assumption :

**A4** For all $k$,

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{\text{mdc}} \max\left[\|g_k\| \min\left[\frac{\|g_k\|}{\beta_k}, \Delta_k\right], |\tau_k| \min[\tau_k^2, \Delta_k^2]\right]$$

where $\kappa_{\text{mdc}} \in (0, 1)$, $\beta_k$ is defined in (4.6) and $\tau_k$ in (4.7).

Instead of requiring A2, we could have supposed conditions (4.11) separately. Furthermore, note that Assumptions A1, A3 and A4 are typical of convergence theory for trust-region methods[(2)] (see [34, Chapter 6]).
For the purpose of our analysis, we shall also consider the following sets of indices :

$$\mathcal{S} = \{k \mid x_{k+1} = x_k + s_k\},$$

the set of *successful iterations*,

$$\mathcal{A} = \{k \mid g_k^+ \text{ is added to the filter }\},$$

the set of *filter iterations*,

$$\mathcal{D} = \{k \mid \rho_k \geq \eta_1\},$$

the set of *sufficient descent iterations*, and

$$\mathcal{N} = \{k \mid \texttt{NONCONVEX} \text{ is set}\},$$

the set of *nonconvex iterations*. Observe that $\mathcal{A} \subseteq \mathcal{S}$, because $g_k^+$ is added to the filter only if the trial point $x_k^+$ is accepted as the new iterate $x_{k+1}$. We also have that

$$\mathcal{S} \cap \mathcal{N} = \mathcal{D} \cap \mathcal{N}, \tag{4.13}$$

since, when the model is nonconvex, the trial point is accepted only if $\rho_k \geq \eta_1$ and the step is within the trust region.
We conclude this section by stating a property of the algorithm that is of crucial importance for the remainder of the convergence analysis.

[(2)] Assumption A3 in [34, Chapter 6] is made over the trust region instead of $\mathbb{R}^n$.

**Lemma 4.1** We have that, for all $k \geq 0$,

$$f(x_0) - f(x_{k+1}) \geq \sum_{\substack{j=0 \\ j \in \mathcal{S} \cap \mathcal{N}}}^{k} [f(x_j) - f(x_{j+1})]. \tag{4.14}$$

**Proof.** Denoting $\mathcal{S} \cap \mathcal{N} = \{k_i\}$, we observe that the definition of $f_{\sup}$ in the algorithm ensures that

$$f(x_{k_i+1}) - f(x_{k_{i+1}}) \geq 0$$

for all $i$. This directly implies that

$$f(x_0) - f(x_{k+1}) = \sum_{j=0}^{k} [f(x_j) - f(x_{j+1})] \geq \sum_{\substack{j=0 \\ j \in \mathcal{S} \cap \mathcal{N}}}^{k} [f(x_j) - f(x_{j+1})].$$

□

### 4.2.2 Convergence to first-order critical points

In this section, we will show that Assumptions A1-A4 ensure that at least one limit point $x^*$ of the sequence $\{x_k\}$ generated by Algorithm 4.1 is a first-order critical point for problem (4.3), that is, it satisfies

$$\nabla_x f(x^*) = 0.$$

The first step of our convergence analysis is to prove that, as long as a first-order critical point is not approached, we do not have infinitely many successful nonconvex iterations in the course of the algorithm. For completeness of the convergence theory, we start by recalling three results from [34, Chapter 6] in order to show that the trust-region radius is bounded away from zero in this case.

The first lemma shows that the error between the objective function and the model decreases quadratically with the trust-region radius. The proof is identical to that of Theorem 6.4.1 in [34] but we now need to make the additional assumption $\|s_k\| \leq \Delta_k$ explicit, instead of being implicit, in this reference, in the definition of a trust-region step.

**Lemma 4.2** Suppose that A1-A3 hold and that $\|s_k\| \leq \Delta_k$. Then we have that

$$|f(x_k + s_k) - m_k(x_k + s_k)| \leq \kappa_{\text{ubh}} \Delta_k^2, \tag{4.15}$$

where $x_k + s_k \in \mathcal{B}_k$ and

$$\kappa_{\text{ubh}} \stackrel{\text{def}}{=} \max[\kappa_{\text{ufh}}, \kappa_{\text{umh}}]. \tag{4.16}$$

**Proof.** Using A1 and A3, we may apply the mean value theorem (cfr Theorem 1.1) on the objective function and its model, and we obtain that

$$f(x_k + s_k) = f(x_k) + s_k^T g_k + \tfrac{1}{2} s_k^T \nabla_{xx}^2 f(\xi_k) s_k \tag{4.17}$$

for some $\xi_k$ in the segment $[x_k, x_k + s_k]$ and that

$$m_k(x_k + s_k) = m_k(x_k) + s_k^T g_k + \tfrac{1}{2} s_k^T \nabla_{xx}^2 m_k(\zeta_k) s_k \tag{4.18}$$

for some $\zeta_k$ in the segment $[x_k, x_k + s_k]$. Subtracting (4.18) from (4.17) and taking absolute values yields that

$$\begin{aligned} |f(x_k + s_k) - m_k(x_k + s_k)| &= \tfrac{1}{2} |s_k^T \nabla_{xx}^2 f(\xi_k) s_k - s_k^T \nabla_{xx}^2 m_k(\zeta_k) s_k| \\ &\leq \tfrac{1}{2} |s_k^T \nabla_{xx}^2 f(\xi_k) s_k| + \tfrac{1}{2} |s_k^T \nabla_{xx}^2 m_k(\zeta_k) s_k| \\ &\leq \tfrac{1}{2} \|s_k\|^2 \|\nabla_{xx}^2 f(\xi_k)\| + \tfrac{1}{2} \|s_k\|^2 \|\nabla_{xx}^2 m_k(\zeta_k)\| \\ &\leq \tfrac{1}{2} (\kappa_{\text{ufh}} + \kappa_{\text{umh}} - 1) \Delta_k^2, \end{aligned}$$

where we have used A1, A2, A3, the triangle and Cauchy-Schwarz inequalities, and the fact that $\|s_k\| \leq \Delta_k$. Thus (4.15) holds with the definition (4.16) of $\kappa_{\text{ubh}}$. □

We now show that an iteration must be successful and the trust-region radius must increase if the current iterate is not first-order critical and the trust-region radius is small enough.

**Lemma 4.3** Suppose that A1-A4 hold and that $\|s_k\| \leq \Delta_k$. Suppose furthermore that $g_k \neq 0$ and that

$$\Delta_k \leq \frac{\kappa_{\text{mdc}} \|g_k\| (1 - \eta_2)}{\kappa_{\text{ubh}}}. \tag{4.19}$$

Then iteration $k$ is very successful, *i.e.* $\rho_k \geq \eta_2$, and

$$\Delta_{k+1} \geq \Delta_k. \tag{4.20}$$

**Proof.** Observe first that the condition $\eta_2 \in (0, 1)$ and the inequality $0 < \kappa_{\text{mdc}} < 1$ together imply that

$$\kappa_{\text{mdc}}(1 - \eta_2) < 1.$$

Thus conditions (4.19), (4.12) and (4.16) imply that

$$\Delta_k < \frac{\|g_k\|}{\beta_k}. \tag{4.21}$$

As a consequence, A4 gives that

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{\text{mdc}}\|g_k\| \min\left[\frac{\|g_k\|}{\beta_k}, \Delta_k\right] = \kappa_{\text{mdc}}\|g_k\|\Delta_k. \tag{4.22}$$

On the other hand, we may apply Lemma 4.2 and deduce from (4.22), (4.15), (4.21) and (4.19) that

$$|\rho_k - 1| = \left|\frac{f(x_k + s_k) - m_k(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)}\right| \leq \frac{\kappa_{\text{ubh}}\Delta_k}{\kappa_{\text{mdc}}\|g_k\|} \leq 1 - \eta_2.$$

Therefore, $\rho_k \geq \eta_2$ and the iteration is very successful. Furthermore, (4.10) ensures that (4.20) holds. □

As a consequence, we obtain that the radius cannot become too small as long as a first-order critical point is not approached, which is crucial for the progress of the algorithm.

**Lemma 4.4** Suppose that A1-A4 hold and that there exists a constant $\kappa_{\text{lbg}} > 0$ such that $\|g_k\| \geq \kappa_{\text{lbg}}$ for all $k$. Then there is a constant $\kappa_{\text{lbd}} > 0$ such that

$$\Delta_k \geq \kappa_{\text{lbd}} \tag{4.23}$$

for all $k$.

**Proof.** Assume that iteration $k$ is the first such that

$$\Delta_{k+1} \leq \gamma_1 \min\left[\Delta_0, \frac{\kappa_{\text{mdc}}\,\kappa_{\text{lbg}}(1 - \eta_2)}{\kappa_{\text{ubh}}}\right] \stackrel{\text{def}}{=} \gamma_1\delta_0. \tag{4.24}$$

This means that the trust-region radius has been decreased at iteration $k$, which in turn implies, from the condition in Step 4 of the algorithm, that $\|s_k\| \leq \Delta_k$. We also have that $\gamma_1\Delta_k \leq \Delta_{k+1}$, and hence that

$$\Delta_k \leq \delta_0 \leq \frac{\kappa_{\text{mdc}}\,\kappa_{\text{lbg}}(1 - \eta_2)}{\kappa_{\text{ubh}}}.$$

Our assumption on the norm of the gradient then implies that (4.19) holds. This and the fact that $\|s_k\| \leq \Delta_k$ thus give that (4.20) is satisfied. But this contradicts the fact that iteration $k$ is the first such that (4.24) holds, and our initial assumption is therefore impossible. This yields the desired conclusion with $\kappa_{\text{lbd}} = \gamma_1 \delta_0$. □

We now prove the crucial result that the number of successful nonconvex iterations must be finite unless a first-order critical point is approached.

**Theorem 4.5** Suppose that A1-A4 hold and that there exists a constant $\kappa_{\text{lbg}} > 0$ such that $\|g_k\| \geq \kappa_{\text{lbg}}$ for all $k$. Then there can only be finitely many successful nonconvex iterations in the course of the algorithm, *i.e.*

$$|\mathcal{S} \cap \mathcal{N}| < +\infty.$$

**Proof.** Suppose, for the purpose of obtaining a contradiction, that there are infinitely many successful nonconvex iterations, which we index by $\mathcal{S} \cap \mathcal{N} = \{k_i\}$. It follows from (4.13) that the algorithm also guarantees that $\rho_k \geq \eta_1$ for all iterations in $\mathcal{S} \cap \mathcal{N}$, which in turn implies, with A4, that, for $k \in \mathcal{S} \cap \mathcal{N}$,

$$\begin{aligned} f(x_k) - f(x_{k+1}) &\geq \eta_1 [m_k(x_k) - m_k(x_k + s_k)] \\ &\geq \eta_1 \, \kappa_{\text{mdc}} \|g_k\| \min\left[\frac{\|g_k\|}{\beta_k}, \Delta_k\right] \\ &\geq \eta_1 \, \kappa_{\text{mdc}} \, \kappa_{\text{lbg}} \min\left[\frac{\kappa_{\text{lbg}}}{\kappa_{\text{umh}}}, \kappa_{\text{lbd}}\right], \end{aligned}$$

where we have used Lemma 4.4, (4.12) and our lower bound on the gradient norm to obtain the last inequality. Combining now this bound with (4.14), we deduce that

$$f(x_0) - f(x_{k+1}) \geq \sum_{\substack{j=0 \\ j \in \mathcal{S} \cap \mathcal{N}}}^{k} [f(x_j) - f(x_{j+1})] \geq \varsigma_k \, \eta_1 \, \kappa_{\text{mdc}} \, \kappa_{\text{lbg}} \min\left[\frac{\kappa_{\text{lbg}}}{\kappa_{\text{umh}}}, \kappa_{\text{lbd}}\right],$$

where $\varsigma_k = |\{1, \ldots, k\} \cap \mathcal{S} \cap \mathcal{N}|$. As we have supposed that there are infinitely many successful nonconvex iterations, we have that

$$\lim_{k \to \infty} \varsigma_k = +\infty,$$

and $[f(x_0) - f(x_{k+1})]$ is unbounded above, which contradicts the fact that the objective

function is bounded below, as stated in (4.11). Our initial assumption must then be false, and the set $\mathcal{S} \cap \mathcal{N}$ of successful nonconvex iterations must be finite. □

We now establish the criticality of the limit point of the sequence of iterates when there are only finitely many successful iterations.

**Theorem 4.6** Suppose that A1-A4 hold and that there are only finitely many successful iterations, *i.e.* $|\mathcal{S}| < +\infty$. Then $x_k = x^*$ for all sufficiently large $k$, and $x^*$ is a first-order critical point, that is

$$\nabla_x f(x^*) = 0.$$

**Proof.** Let $k_0$ be the index of the last successful iterate. Then $x^* = x_{k_0+1} = x_{k_0+j}$ for all $j > 0$. Furthermore,

$$\rho_{k_0+j} < \eta_1 \quad \text{for all} \quad j > 1. \tag{4.25}$$

Now observe that `RESTRICT` is set by Algorithm 4.1 in the course of every unsuccessful iteration. This flag must thus be set at the beginning of every iteration of index $k_0 + j + 1$ for $j > 0$. As a consequence, $\|s_{k_0+j+2}\| \leq \Delta_{k_0+j+2}$ for all $j > 0$. This, (4.25) and the mechanism of Step 4 of the algorithm then imply that

$$\lim_{k \to \infty} \Delta_k = 0. \tag{4.26}$$

Assume now, for the purpose of establishing a contradiction, that $\|g_{k_0+1}\| \geq \varepsilon$ for some $\varepsilon > 0$. Then Lemma 4.4 implies that (4.26) is impossible and we deduce that

$$\|g_{k_0+j}\| = 0$$

for all $j > 0$ and we obtain the desired conclusion. □

Having proved the desired convergence property for the case where $\mathcal{S}$ is finite, we restrict our attention, for the rest of this section, to the case where there are infinitely many successful iterations, *i.e.* $|\mathcal{S}| = +\infty$. We first investigate what happens if infinitely many values are added to the filter in the course of the algorithm, *i.e.* $|\mathcal{A}| = +\infty$.

**Theorem 4.7** Suppose that A1-A4 hold and that $|\mathcal{A}| = |\mathcal{S}| = +\infty$. Then

$$\liminf_{k\to\infty} \|g_k\| = 0. \tag{4.27}$$

In other words, there exists a limit point $x^*$ of the sequence $\{x_k\}$ generated by the algorithm which is a first-order critical point for problem (4.3).

**Proof.** Assume, for the purpose of obtaining a contradiction, that, for all $k$ large enough,

$$\|g_k\| \geq \kappa_{\text{lbg}} \tag{4.28}$$

for some $\kappa_{\text{lbg}} > 0$ and define $\{k_i\} = \mathcal{A}$. The bound (4.28) and Theorem 4.5 then imply that $|\mathcal{S} \cap \mathcal{N}|$ is finite and therefore that the filter is no longer reset to the empty set for $k$ sufficiently large. Moreover, since our assumptions imply that $\{\|g_{k_{i+1}}\|\}$ is bounded above and away from zero, there must exist a subsequence $\{k_\ell\} \subseteq \{k_{i+1}\}$ such that

$$\lim_{\ell\to\infty} g_{k_\ell} = g_\infty \qquad \text{with} \quad \|g_\infty\| \geq \kappa_{\text{lbg}}. \tag{4.29}$$

By definition of $\{k_\ell\}$, $x_{k_\ell}$ is acceptable for the filter in each iteration $\ell - 1$. This implies, since the filter is not reset for $\ell$ large enough, that, for each $\ell$ sufficiently large, there exists an index $j_\ell \in \{1, \ldots, n\}$ such that

$$|g_{k_\ell, j_\ell}| - |g_{k_{\ell-1}, j_\ell}| < -\gamma_g \|g_{k_{\ell-1}}\|. \tag{4.30}$$

But (4.28) implies that $\|g_{k_{\ell-1}}\| \geq \kappa_{\text{lbg}}$ for all $\ell$ sufficiently large. Hence we deduce from (4.30) that

$$|g_{k_\ell, j_\ell}| - |g_{k_{\ell-1}, j_\ell}| < -\gamma_g \kappa_{\text{lbg}}$$

for all $\ell$ sufficiently large. But the left-hand side of this inequality tends to zero when $\ell$ tends to infinity because of (4.29), yielding the desired contradiction. Hence (4.27) holds. □

Consider now the case where the number of iterates added to the filter in the course of the algorithm is finite, *i.e.* $|\mathcal{A}| < +\infty$.

**Theorem 4.8** Suppose that A1-A4 hold and that $|\mathcal{S}| = +\infty$ but $|\mathcal{A}| < +\infty$. Then (4.27) holds.

**Proof.** Assume, again for the purpose of obtaining a contradiction, that (4.28) holds for all $k$ large enough and for some $\kappa_{\text{lbg}} > 0$. The finiteness of $|\mathcal{A}|$ then implies that $\rho_k \geq \eta_1$ and that $\|s_k\| \leq \Delta_k$ for all $k \in \mathcal{S}$ sufficiently large. If we define $\bar{\varsigma}_{p,k} = |\{p, \ldots, k\} \cap \mathcal{S}|$, we then obtain that

$$f(x_p) - f(x_{k+1}) = \sum_{\substack{j=p \\ j \in \mathcal{S}}}^{k} [f(x_j) - f(x_{j+1})] \geq \bar{\varsigma}_{p,k}\, \eta_1\, \kappa_{\text{mdc}}\, \kappa_{\text{lbg}} \min\left[\frac{\kappa_{\text{lbg}}}{\kappa_{\text{umh}}}, \kappa_{\text{lbd}}\right],$$

for $p$ and $k$ sufficiently large, where, as above, we used A4, (4.12), (4.23) and (4.28) to derive the inequality. But $\bar{\varsigma}_{p,k}$ tends to infinity with $k$ for a fixed $p$ sufficiently large since $|\mathcal{S}|$ is infinite, and we again derive a contradiction from the fact that $f(x_{k+1})$ then becomes unbounded below. The limit (4.27) then follows. □

By the last two theorems, we have that at least one of the limit points of the sequence of iterates generated by the algorithm satisfies the first-order necessary condition (see Section 2.2.1). As the following counter-example shows, it is not possible to obtain a stronger result in Theorems 4.7 and 4.8 such as

$$\lim_{k \to \infty} \|g_k\| = 0$$

without modifying the algorithm.

### 4.2.3 Analysis of a counter-example

This counter-example has been suggested by Nick Gould. Consider the objective function

$$f(x) = x^3(3x - 4),$$

which is represented in Figure 4.2.

We can see that this function has a *degenerate* critical point at $x = 0$, *i.e.*, both its first and second derivatives vanish, and its global minimizer at $x = 1$. We will show that it is possible for Algorithm 4.1 to construct iterates for which

$$x_{2k} = -\tfrac{1}{2}^k \quad \text{and} \quad x_{2k+1} = \tfrac{5}{4},$$

for $k = 0, 1, 2, \ldots$ Clearly there are two limit points, $x_{\text{L}}^* = 0$ and $x_{\text{R}}^* = \frac{5}{4}$, but only the first is critical.

Let $\Delta_0 > 3$, and suppose that $\gamma_g < \frac{1}{2}$ and that the trust-region updating scheme (2.7) is specifically

$$\Delta_{k+1} = \begin{cases} \frac{1}{2}\Delta_k & \text{if } \rho_k < \eta_1, \\ \Delta_k & \text{if } \eta_1 \leq \rho_k < \eta_2 \text{ and} \\ 2\Delta_k & \text{if } \eta_2 \leq \rho_k. \end{cases} \tag{4.31}$$

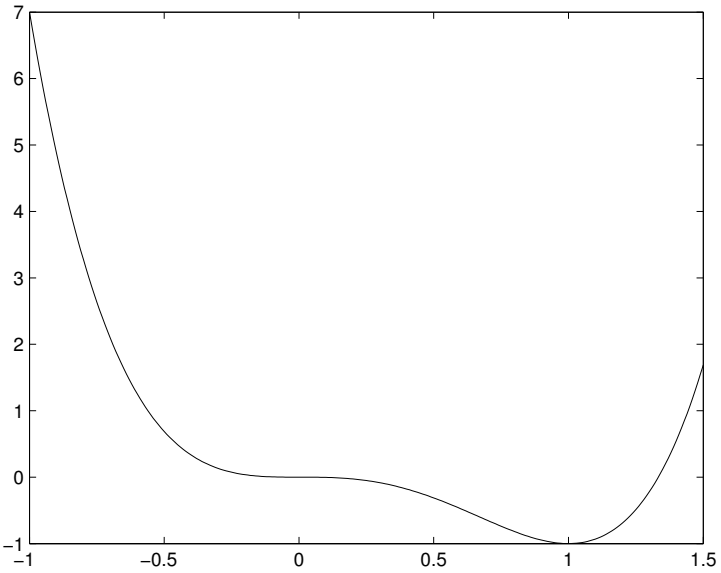


Figure 4.2: The plot of the function $f(x) = x^3(3x - 4)$

Now suppose that

$$\mathcal{F} = \{f'(x_{2k})\} \equiv \{-12(1 + \tfrac{1}{2}^k)\tfrac{1}{2}^{2k}\} \text{ and } \Delta_{2k} > 3. \tag{4.32}$$

We then show that the above iteration is possible for Algorithm 4.1, and that (4.32) will persist.

Consider first $x_{2k} = -\frac{1}{2}^k$, and the convex model

$$m_{2k}(x_{2k} + s) = f(x_{2k}) + sf'(x_{2k}) + \tfrac{1}{2}s^2 h_{2k},$$

where

$$h_{2k} = -\frac{f'(x_{2k})}{\frac{5}{4} - x_{2k}} > 0.$$

Then the unconstrained global minimizer of $m_{2k}$ is $s_{2k} = \frac{5}{4} - x_{2k}$, and $s_{2k}$ will sufficiently reduce the model within the trust region since $\Delta_{2k} > 3 > \frac{5}{4} + (\frac{1}{2})^k$. In Figure 4.3, the dashed line represents the model built at $x_0$ and the star its minimum.

Moreover,

$$\begin{aligned} m_{2k}(x_{2k}) - m_{2k}(x_{2k} + s_{2k}) &= -s_{2k}f'(x_{2k}) - \tfrac{1}{2}s_{2k}^2 h_{2k} \\ &= \tfrac{1}{2}\frac{(f'(x_{2k}))^2}{h_{2k}} \\ &= -\tfrac{1}{2}(\tfrac{5}{4} - x_{2k})f'(x_{2k}) \to 0 \end{aligned}$$

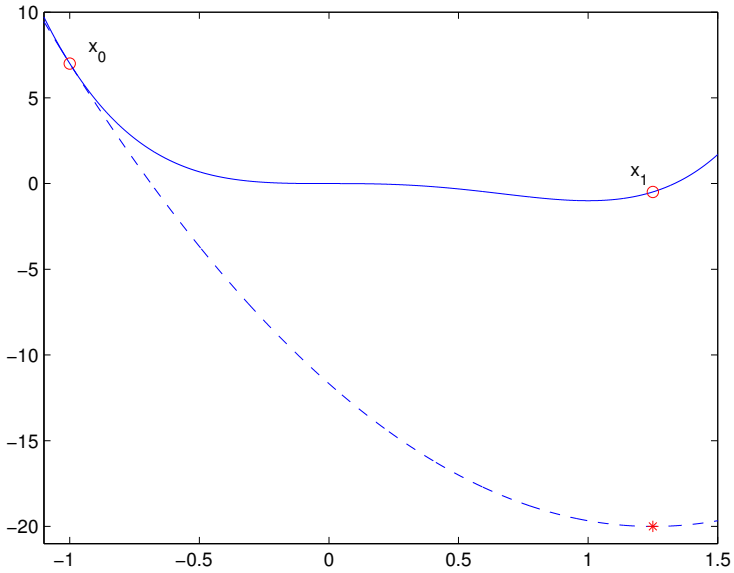


Figure 4.3: The first two iterates and the model around $x_0$

while

$$f(x_{2k}) - f(x_{2k} + s_{2k}) = f(x_{2k}) - f\left(\frac{5}{4}\right) > f(0) - f\left(\frac{5}{4}\right) = \frac{125}{256} > 0$$

and thus

$$\rho_{2k} \geq \eta_2 \tag{4.33}$$

for large enough $k$. The trial point $x_{2k} + s_{2k}$ is not acceptable for the filter since its gradient is

$$f'\left(\frac{5}{4}\right) = \frac{75}{16} \gg f'(x_{2k}),$$

but it is an acceptable point because the trust-region bound is inactive and because of (4.33). Thus $x_{2k+1} = x_{2k} + s_{2k} = \frac{5}{4}$, while (4.31) and (4.33) ensure that $\Delta_{2k+1} = 2\Delta_{2k}$.

Now consider $x_{2k+1} = \frac{5}{4}$, and the convex model

$$m_{2k+1}(x_{2k+1} + s) = f(x_{2k+1}) + sf'(x_{2k+1}) + \tfrac{1}{2}s^2 h_{2k+1},$$

where

$$h_{2k+1} = \frac{f'(x_{2k+1})}{x_{2k+1} + \frac{1}{2}^{k+1}} > 0.$$

As before, the unconstrained global minimizer of $m_{2k+1}$ is $s_{2k+1} = -x_{2k+1} - \frac{1}{2}^{k+1}$, and $s_{2k+1}$ will sufficiently reduce the model within the trust region since $\Delta_{2k+1} > 6 > \frac{5}{4} + (\frac{1}{2})^k$. Figure 4.4 illustrates the first three iterates.

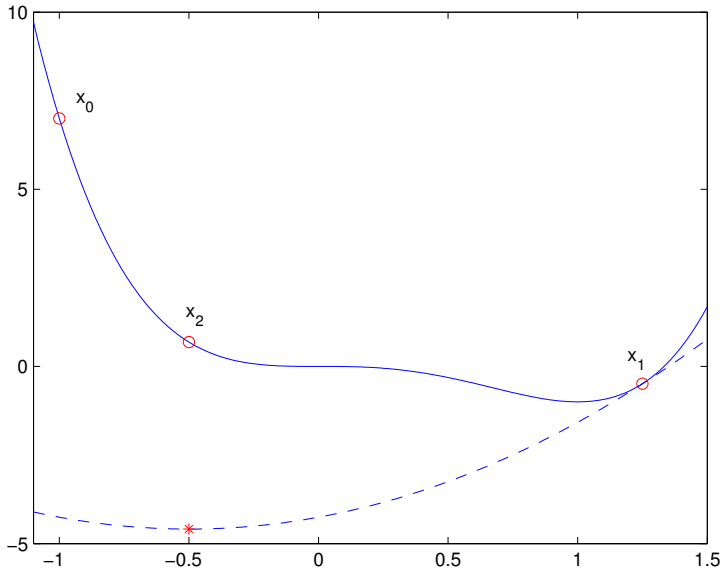


Figure 4.4: The first three iterates and the model around $x_1$

Remark that, although

$$\begin{aligned} m_{2k+1}(x_{2k+1}) - m_{2k+1}(x_{2k+1}+s_{2k+1}) &= \tfrac{1}{2}\tfrac{(f'(x_{2k+1}))^2}{h_{2k+1}} \\ &= \tfrac{1}{2}(x_{2k+1}+\tfrac{1}{2}^{k+1})f'(x_{2k+1}) > 0 \end{aligned}$$

and

$$f(x_{2k+1}) - f(x_{2k+1}+s_{2k+1}) < f\left(\frac{5}{4}\right) - f(0) = -\frac{125}{256} < 0$$

and hence

$$\rho_{2k+1} < 0, \tag{4.34}$$

$x_{2k+1}+s_{2k+1} = -\frac{1}{2}^{k+1}$ is acceptable for the filter since it is easy to check that

$$|f'(x_{2k+1}+s_{2k+1})| = |f'(-\tfrac{1}{2}^{k+1})| < \tfrac{1}{2}|f'(x_{2k})|.$$

Hence $x_{2k+2} = x_{2k+1}+s_{2k+1} = -\frac{1}{2}^{k+1}$. Moreover (4.31) and (4.34) imply that $f'(x_{2k+2})$ replaces $f'(x_{2k})$ in the filter, and that $\Delta_{2k+2} = \frac{1}{2}\Delta_{2k+1} = \Delta_{2k}$, and thus that (4.32) persists. If we continue, the iterates will converge towards two limit points, namely $0$ and $\frac{5}{4}$, as suggested on the plots of Figure 4.5.

It is unclear how to enforce the property that all limit points are first-order critical without adversely affecting the algorithm's numerical behaviour. We have considered not allowing filter

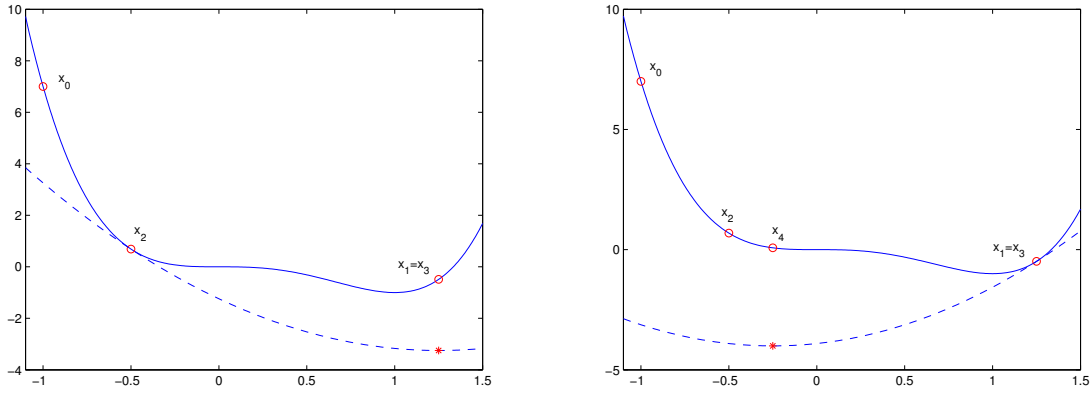


Figure 4.5: The next iterates

iterations when the ratio between the current gradient norm and the smallest gradient norm found so far exceeds some prescribed (large) constant. While such a modification does not appear to affect the results of our numerical experiments, we have been unable to show that the modification yields the desired conclusion. Since we believe that the likelihood of the algorithm converging to more than a single limit point is very small (as with every trust-region method we are aware of), the issue really is of mostly theoretical interest.

### 4.2.4 Convergence to second-order critical points

We thus pursue our analysis by examining convergence to second-order critical points under the assumption that there is only one limit point. As in [34], we also assume the following :

**A5** The matrix $H_k$ is arbitrarily close to $\nabla^2_{xx} f(x_k)$ whenever a first-order critical point is approached, *i.e.*

$$\lim_{k\to\infty} \|\nabla^2_{xx} f(x_k) - H_k\| = 0 \quad \text{whenever} \quad \lim_{k\to\infty} \|g_k\| = 0.$$

Remark that $h_{2k}$ tends to zero, and thus that A5 holds in the counter-example of Section 4.2.3. Given this assumption on the quality of the model of the objective function, we are then able to derive the following theorem.

**Theorem 4.9** Suppose that A1-A5 hold and that the complete sequence of iterates $\{x_k\}$ produced by Algorithm 4.1 converges to the unique limit point $x^*$. Then $x^*$ is a second-order critical point.

**Proof.** Our proof is strongly inspired by Theorem 6.6.4 of [34]. Observe that our previous results imply that

$$g(x^*) = 0. \tag{4.35}$$

For the purpose of deriving a contradiction, assume now that

$$\tau_* \stackrel{\text{def}}{=} \lambda_{\min}(\nabla^2_{xx} f(x^*)) < 0. \tag{4.36}$$

Then, using A5 and (4.35), we deduce that there exists an index $k_0$ such that, for $k \geq k_0$,

$$\lambda_{\min}(H_k) < \tfrac{1}{2}\tau_* < 0,$$

and, consequently, that $k \in \mathcal{N}$ and

$$\|s_k\| \leq \Delta_k \tag{4.37}$$

for $k \geq k_0$. Our sufficient decrease condition given in A4 then ensures that, for $k \geq k_0$,

$$m_k(x_k) - m_k(x_k + s_k) \geq \tfrac{1}{2}\kappa_{\text{mdc}}|\tau_*| \min[\tfrac{1}{4}\tau_*^2, \Delta_k^2]. \tag{4.38}$$

Consider now the ratio of achieved versus predicted reduction $\rho_k$ in the case where $\Delta_k \leq \frac{1}{2}|\tau_*|$. This, (4.38) and (4.37) imply that

$$m_k(x_k) - m_k(x_k + s_k) \geq \tfrac{1}{2}\kappa_{\text{mdc}}|\tau_*|\Delta_k^2 \geq \tfrac{1}{2}\kappa_{\text{mdc}}|\tau_*|\|s_k\|^2 \tag{4.39}$$

for $k \geq k_0$. Using the mean value theorem and the Cauchy-Schwarz inequality successively, we obtain that, for some $\xi_k$ in the segment $[x_k, x_k + s_k]$,

$$\begin{aligned}
|\rho_k - 1| &= \left| \frac{f(x_k + s_k) - m_k(x_k + s_k)}{m_k(x_k) - m_k(x_k + s_k)} \right| && \text{(4.40a)} \\
&\leq 2\, \frac{|s_k^T \nabla^2_{xx} f(\xi_k) s_k - s_k^T H_k s_k|}{\kappa_{\text{mdc}}|\tau_*|\|s_k\|^2} && \text{(4.40b)} \\
&\leq \frac{2}{\kappa_{\text{mdc}}|\tau_*|}\, \|\nabla^2_{xx} f(\xi_k) - H_k\| && \text{(4.40c)}
\end{aligned}$$

for $k \geq k_0$. Since $\|\xi_k - x_k\| \leq \|s_k\| \leq \Delta_k$ for $k \geq k_0$, A1, (4.35) and A5 imply that (4.40c) must be arbitrarily small for $\Delta_k$ sufficiently small and $k$ sufficiently large. Thus, there must exist an index $k_1 \geq k_0$ and a $\delta_1 \in (0, \frac{1}{2}|\tau_*|]$ such that

$$\rho_k \geq \eta_2 \;\text{ for all }\; k \geq k_1 \;\text{ such that }\; \Delta_k \leq \delta_1.$$

As a consequence, each iteration where these two conditions hold must be very successful and the algorithm then guarantees that $\Delta_{k+1} \geq \Delta_k$. This and the inequality $\gamma_1\delta_1 < \delta_1 \leq \frac{1}{2}|\tau_*|$ in turn imply that

$$\Delta_k \geq \min[\gamma_1\delta_1, \Delta_{k_0}] \stackrel{\text{def}}{=} \delta_2 \tag{4.41}$$

for all $k \geq k_1$. For every successful iteration $k \geq k_1$, we then obtain from (4.38) that

$$f(x_k) - f(x_{k+1}) \geq \tfrac{1}{2}\eta_1 \kappa_{\text{mdc}} |\tau_*| \min[\tfrac{1}{4}\tau_*^2, \delta_2^2] > 0.$$

Remembering now that $k \in \mathcal{N}$ for $k \geq k_1$ (and thus that $|\mathcal{N}| = \infty$), we obtain from (4.14) that $|\mathcal{S} \cap \mathcal{N}|$, and hence $|\mathcal{S}|$, must be finite, which in turn implies that the trust-region radius tends to zero. But this contradicts (4.41). Hence our initial assumption (4.36) must be false and the proof is complete. □

## 4.3 Conclusion

We have presented in this chapter a novel filter-trust-region algorithm for solving unconstrained optimization problems. We have shown, under standard assumptions in trust-region framework, that our algorithm produces at least a first-order critical point, irrespective of the chosen starting point. Under mild additional conditions, we have also proved that convergence of the complete sequence of iterates can only occur to a second-order critical point. The performance of this algorithm will be discussed in the next chapter.

# Chapter 5

# Numerical results

This chapter will be devoted to the practical performance of the algorithm presented in Section 4.1. The implementation of this algorithm is called FILTRUNC (FILter Trust-Region algorithm for UNConstrained optimization) and is based on the code FILTRANE [76] from the GALAHAD library.
We will start this chapter by considering the testing environment in which our numerical tests are performed. Section 5.2 will briefly introduce performance profiles, a recent tool for the assessment and comparison of numerical packages, while in Section 5.3 some practical aspects of the implementation of our algorithm are discussed. Numerical comparisons of our software with LANCELOT-B will be given in Section 5.4, attesting the reliability and the efficiency of our method, and several algorithmic variants will also be discussed and compared in this section. We will close the chapter with a short conclusion on the numerical experience.

## 5.1 Testing environment

In order to assess the potential of our method in solving unconstrained nonlinear optimization problems, we have tested our implementation on the CUTEr (Constrained and Unconstrained Testing Environment, revisited) set of test problem [69]. This collection of nonlinear problems has originally been built by Bongartz et al. [13] and is now widely used by the community of optimization researchers interested in the development of new software. The source and documentations for this project are publicly available at the following address :

`http://cuter.rl.ac.uk/cuter-www/`

One of the most important parts of the CUTEr environment is its decoder, called SifDec. In fact, the problems of this test set are formulated in the Standard Input Fortran (SIF)[(1)]. Once

[(1)]Note that a version of these problems translated in AMPL is available.

they have been decoded and translated into a set of Fortran routines, these files may be linked to the optimization code and provide tools for it, like, for example, subroutines allowing to evaluate the objective and constraint functions, or their derivatives. It is also possible to obtain the number of variables involved or statistics concerning function evaluation, CPU time and so on.

With the `select` utility, we have extracted from the CUTEr collection problems sharing the following features (where * means "anything goes") :

| | | |
|---|---|---|
| Objective function type | : | * |
| Constraints type | : | U (no constraint), X (fixed variables only) |
| Regularity | : | R (regularity) |
| Degree of available derivatives | : | 2 (analytical second derivatives) |
| Problem interest | : | * |
| Explicit internal variables | : | * |
| Number of variables | : | * |
| Number of constraints | : | * |

Table 5.1: The unconstrained problem selection

Note that we include problems with fixed variables. These choices result in 160 problems. Our numerical results are thus obtained by running our implementation of Algorithm 4.1 on this set of 159 unconstrained[(2)] problems from the CUTEr collection.

Problem names and their dimensions[(3)] are detailed in Table 5.3, while Table 5.2 gives the distribution of problem dimension in the CUTEr collection. It can be seen that the size of the problems varies significantly. The user is allowed to modify the dimension of many of these problems.

| # of free variables | # of problems from the CUTEr set |
|---|---|
| $2 \leq n \leq 9$ | 61 |
| $10 \leq n \leq 99$ | 15 |
| $100 \leq n \leq 999$ | 16 |
| $1000 \leq n \leq 9999$ | 57 |
| $10000 \leq n \leq 20000$ | 10 |

Table 5.2: The distribution of problem dimension in the CUTEr collection

[(2)] We excluded problem `BROYDN7D` because of its multiple local minima.

[(3)] The number of free variables.

| Problem | $n$ | Problem | $n$ | Problem | $n$ |
|---|---|---|---|---|---|
| AIRCRFTB | 5 | DQRTIC | 5000 | OSBORNEA | 5 |
| ALLINITU | 4 | EDENSCH | 10000 | OSBORNEB | 11 |
| ARGLINA | 200 | EG2 | 1000 | PALMER1C | 8 |
| ARGLINB | 200 | EIGENALS | 2550 | PALMER1D | 7 |
| ARGLINC | 200 | EIGENBLS | 2550 | PALMER2C | 8 |
| ARWHEAD | 5000 | EIGENCLS | 2652 | PALMER3C | 8 |
| BARD | 3 | ENGVAL1 | 10000 | PALMER4C | 8 |
| BDQRTIC | 5000 | ENGVAL2 | 2 | PALMER5C | 6 |
| BEALE | 2 | ERRINROS | 50 | PALMER6C | 8 |
| BIGGS3 | 3 | EXPFIT | 2 | PALMER7C | 8 |
| BIGGS5 | 5 | EXTROSNB | 1000 | PALMER8C | 8 |
| BIGGS6 | 6 | FMINSRF2 | 5625 | PARKCH | 15 |
| BOX2 | 2 | FMINSURF | 49 | PENALTY1 | 1000 |
| BOX3 | 3 | FREUROTH | 5000 | PENALTY2 | 200 |
| BRKMCC | 2 | GENROSE | 500 | PENALTY3 | 200 |
| BROWNAL | 200 | GROWTHLS | 3 | POWELLSG | 5000 |
| BROWNBS | 2 | GULF | 3 | POWER | 100 |
| BROWNDEN | 4 | HAIRY | 2 | QUARTC | 5000 |
| BRYBND | 5000 | HATFLDD | 3 | RAYBENDL | 2046 |
| CHAINWOO | 4000 | HATFLDE | 3 | RAYBENDS | 2046 |
| CHNROSNB | 50 | HEART6LS | 6 | ROSENBR | 2 |
| CLIFF | 2 | HEART8LS | 8 | S308 | 2 |
| CLPLATEA | 10100 | HELIX | 3 | SBRYBND | 500 |
| CLPLATEB | 4970 | HIELOW | 3 | SCHMVETT | 5000 |
| CLPLATEC | 4970 | HILBERTA | 2 | SCOSINE | 5000 |
| COSINE | 10000 | HILBERTB | 10 | SCURLY10 | 100 |
| CRAGGLVY | 5000 | HIMMELBB | 2 | SCURLY20 | 100 |
| CUBE | 2 | HIMMELBF | 4 | SCURLY30 | 100 |
| CURLY10 | 10000 | HIMMELBG | 2 | SENSORS | 100 |
| CURLY20 | 10000 | HIMMELBH | 2 | SINEVAL | 2 |
| CURLY30 | 1000 | HYDC20LS | 99 | SINQUAD | 10000 |
| DECONVU | 61 | JENSMP | 2 | SISSER | 2 |
| DENSCHNA | 2 | KOWOSB | 4 | SNAIL | 2 |
| DENSCHNB | 2 | LIARWHD | 5000 | SPARSINE | 5000 |
| DENSCHNC | 2 | LMINSURF | 5329 | SPARSQUR | 10000 |
| DENSCHND | 3 | LOGHAIRY | 2 | SPMSRTLS | 4900 |
| DENSCHNE | 3 | MANCINO | 100 | SROSENBR | 5000 |
| DENSCHNF | 2 | MARATOSB | 2 | SSC | 4900 |
| DIXMAANA | 9000 | MEXHAT | 2 | STRATEC | 10 |
| DIXMAANB | 9000 | MEYER3 | 3 | TESTQUAD | 5000 |
| DIXMAANC | 9000 | MINSURF | 36 | TOINTGOR | 50 |
| DIXMAAND | 9000 | MOREBV | 5000 | TOINTGSS | 5000 |
| DIXMAANE | 9000 | MSQRTALS | 1024 | TOINTPSP | 50 |
| DIXMAANF | 9000 | MSQRTBLS | 1024 | TOINTQOR | 50 |
| DIXMAANG | 9000 | NCB20 | 5010 | TQUARTIC | 5000 |
| DIXMAANH | 9000 | NCB20B | 5000 | TRIDIA | 5000 |
| DIXMAANI | 9000 | NLMSURF | 5329 | VARDIM | 200 |
| DIXMAANJ | 9000 | NONCVXU2 | 5000 | VAREIGVL | 50 |
| DIXMAANK | 9000 | NONCVXUN | 5000 | VIBRBEAM | 8 |
| DIXMAANL | 9000 | NONDIA | 5000 | WATSON | 12 |
| DIXON3DQ | 10000 | NONDQUAR | 5000 | WOODS | 10000 |
| DJTL | 2 | NONMSQRT | 100 | YFITU | 3 |
| DQDRTIC | 5000 | ODC | 4900 | ZANGWIL2 | 2 |

Table 5.3: The test problems and their dimension

All experimental tests of this chapter were performed in double precision on a workstation with a 3.2 GHz Pentium IV biprocessor and 2 Gbytes of memory under Suse Professional 9.0 Linux and the Lahey Fortran compiler (version L6.10a) with default options. All attempts to solve the test problems were limited to a maximum of 1000 iterations or 1 hour of CPU time. In some situations where problems are solved very quickly, comparative CPU time results may be unreliable because these timings may be dominated by noise. Note that throughout this work, the variability of CPU time for small times is taken into account by repeatedly solving the same problem until a threshold of ten seconds is exceeded and then taking the average time per solve.

## 5.2 Performance profiles

In order to compare different variants of FILTRUNC or our code versus another solver, we will use *performance profiles* proposed by Dolan and Moré [43]. The goal of these profiles, which are now quite useful in benchmarking and for comparisons between softwares, is to capture in a single plot a summary of both efficiency and robustness of several methods or algorithmic variants on a set of test problems. The graph has as many curves as the number of codes we want to compare and each curve gives the efficiency of each code with respect to the other ones.

Suppose that we want to compare a set of solvers $\mathcal{S}$, such as different solvers or several algorithmic options for one solver, on a set of test problems $\mathcal{P}$. Let $q_{p,s}$ denote the quantities we want to compare for problem $p$ and run $s$. These performance measures may be, for instance, iteration count, number of function evaluations or computing time. For these latter statistics, the smaller the value, the better the considered variant. The performance ratio is defined as

$$r_{p,s} \stackrel{\text{def}}{=} \frac{q_{p,s}}{\min\{q_{p,s} \mid s \in \mathcal{S}\}},$$

in order to compare the performance of solver $s$ on problem $p$ with the best performance by any solver in $\mathcal{S}$ on $p$. Note that if the run $s$ has failed on problem $p$, we set the ratio $r_{p,s}$ to infinity. Then, we define the performance profile for each solver $s$ as

$$p_s(\sigma) \stackrel{\text{def}}{=} \frac{\text{card}(p \in \mathcal{P} \mid r_{p,s} \leq \sigma)}{\text{card}(\mathcal{P})}, \qquad \sigma \geq 1.$$

So performance profiles give, for every $\sigma \geq 1$, the proportion $p(\sigma)$ of test problems on which each considered code has a performance within a factor $\sigma$ of the best. For example, the value $p_s(1)$ is the probability that the solver or algorithmic option $s$ is the best. $p_s(2)$ gives the percentage of test problems for which code $s$ is within a factor of 2 of the best. And the limit

$$\lim_{\sigma \to \infty} p_s(\sigma)$$

gives the fraction of test problems of $\mathcal{P}$ for which the code $s$ succeeded. As a consequence, the values on the left of the plot represent the efficiency of each solver and the values on the right give information about the robustness of the solvers. So visually speaking, the "best" solver is the highest on the plot.

## 5.3 Practical aspects

We now consider some practical details about the implementation of Algorithm 4.1. Note that, in each case, the starting point supplied with the problem in the CUTEr collection has been used. While there exist specific strategies for choosing the initial trust-region radius (see, for example, Sartenaer [113]), we have chosen here to start with $\Delta_0 = 1$. The algorithm also depends on parameters that are defined in Step 0 of Algorithm 4.1, in our implementation, they have been set to the following values

$$\gamma_1 = 0.0625, \quad \gamma_2 = 0.25, \quad \gamma_3 = 2.0, \quad \eta_1 = 0.01, \quad \eta_2 = 0.9,$$

and

$$\gamma_g = \min\left[0.001, \frac{1}{2\sqrt{n}}\right].$$

We also have to choose a procedure to compute the next trial point. So, at each iteration, the step $s_k$ is computed by approximately minimizing the model $m_k(x_k + s)$ within the trust region by using the Generalized Lanczos Trust-Region (`GLTR`) algorithm proposed by Gould, Lucidi, Roma and Toint [66] as implemented in the GALAHAD library [70]. This subroutine is built with the aim of finding a better approximation to the solution of the trust-region subproblem (2.18) than that obtained by the Steihaug-Toint algorithm. This latter strategy, proposed separately by Steihaug [118] and Toint [119] and widely-used for large-scale problems, consists of following the path of conjugate-gradient iterates. The iterates generated by the conjugate-gradient algorithm (see Section 2.3.4) are thus used until either they leave the trust region or a negative curvature is detected. In both cases, the last conjugate-gradient step is truncated on the trust-region boundary. So one stops because either convergence or trust-region boundary is reached. The `GLTR` code allows to continue the process when the boundary has been encountered. It is based on solving the subproblem within a subspace defined by the Krylov space[(4)] generated by the conjugate-gradient and Lanczos methods (see [34, Chapter 5]). We refer the reader to the paper of Gould et al. [66] for further details. We have used this method without preconditioning. Some large-scale optimization softwares use this routine as a subproblem

[(4)]The *Krylov* subspace of degree $j$ for $r_0$ is defined as $\mathcal{K}(H, r_0, j) \stackrel{\text{def}}{=} \text{span}\,\{r_0, Hr_0, H^2r_0, \ldots, H^jr_0\}$.

solver (see Byrd, Gould, Nocedal and Waltz [17] and Gould, Orban and Toint [70]).
The procedure is terminated here at the first $s$ for which

$$\|\nabla_x m_k(x_k + s)\| \leq \min\left[0.01, \max(\|\nabla_x m_k(x_k)\|, \sqrt{\varepsilon_M})\right] \quad \|\nabla_x m_k(x_k)\|, \tag{5.1}$$

where $\varepsilon_M$ is the machine precision. This stopping criterion allows an adaptive accuracy in the solution of the trust-region subproblem. It imposes the subproblem to be solved more accurately when gradients are small. Also note that the step obtained by this procedure guarantees the Cauchy point condition

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{\text{mdc}} \|g_k\| \min\left[\frac{\|g_k\|}{\beta_k}, \Delta_k\right],$$

where $\kappa_{\text{mdc}} \in (0, 1)$ and $\beta_k$ is defined in (4.6).

As already mentioned in the previous chapter, the step might be recomputed within the trust region if negative curvature was discovered for the model after having computed a step outside the trust region. This can be achieved by re-entering the `GLTR` subroutine with the new trust-region radius but with all other data unchanged. The subroutine determines the best solution within the Krylov space investigated in the previous minimization (see Gould et al. [66] for more details).

Two particular variants of our algorithm have been tested. The first one, called *filter*, is the algorithm as described in Section 4.1, where exact first and second derivatives are used.
In our algorithm, we choose

$$f_{\text{sup}} = \min(10^6 |f(x_0)|, f(x_0) + 1000)$$

at Step 0.
Based on practical experience (see Section 2.4.3.2 in Gould and Toint [76]), we also impose that

$$\|s_k\| \leq 1000\ \Delta_k \tag{5.2}$$

at all iterations following the first one at which a restricted step was taken. Note that, in Step 1 of Algorithm 4.1, we have to compute a *finite* step $s_k$; otherwise it could happen that the step is infinite if, for instance, the flag `RESTRICT` is unset and the model is linear, *i.e.* $H_k = 0$. In practice, in order to prevent such a situation, we impose a large upper bound on the step, that is

$$\|s_k\| \leq \kappa_\Delta\ \Delta_k,$$

where $\kappa_\Delta$ is the factor by which the trust region is enlarged on unrestricted steps. This factor is initially put to $10^{20}$ until a first restricted step has occurred, afterwards this factor is put to $10^3$ (cfr (5.2)).

Another feature of our filter variant is that we consider the values of the $g(x_k)$ themselves, instead of their absolute values, in the filter test acceptance mechanism. A consequence of this is that the filter is no longer restricted to the positive orthant, and trial points are potentially accepted more often as the new filter acceptance condition is weaker than (4.9). It is interesting to note that the extension of our convergence theory to the use of *unsigned filter entries* (which means that filter entries can take positive or negative values) is simple and straightforward[(5)]. We will present a comparison between variants with unsigned and signed filter entries in Section 5.4.4.

Finally, as already said in the previous chapter, dominated filter points are, in our algorithm, always removed from the filter. We would like to point out that, while the removal of points dominated by a new entry in the filter helps in keeping filter storage low and the amount of comparisons in checking for filter acceptability reasonable, it is not directly relevant to the theory. It is clear that if a point $x$, which dominates a point $y$, is dominated by a new point, say $a$, the point $y$ is also dominated by $a$. We could consider not to remove dominated filter points in our algorithm. In this case, the filter acceptance test would be a bit slower and the inclusion of a new point in the filter a bit faster.

The second algorithmic variant we have considered is the *pure trust-region* variant, which is the same algorithm with the exception that no trial point is ever accepted for the filter and the flag `RESTRICT` is always set, hence trial points are always restricted to the trust region. So, in this version, the filter mechanism is not invoked to decide whether or not a trial point is accepted and it is then similar to a usual monotone trust-region method (see Section 2.3.3 or [34, Chapter 6]).

Note that the Euclidean norm is used to define the trust region for both variants.

As said in Section 4.1, our algorithm stops if the norm of the gradient falls below some tolerance, that is

$$\|\nabla_x f(x_k)\| \leq 10^{-6}\sqrt{n}, \tag{5.3}$$

and the flag `NONCONVEX` is unset. But, in practice, the algorithm will not necessarily detect nonconvexity: our implementation relies on detecting negative curvature in the embedded Krylov subspaces that are used in the `GLTR` subroutine. What the theory in Section 4.2 says is

[(5)]The proof of Theorem 4.7 can be easily derived for the filter acceptance condition with unsigned entries.

that, if negative curvature can be detected reliably, then the convergence to a second-order critical point can be proved. Some difficulties with the termination of the algorithm could thus possibly occur, in the sense that the algorithm might, in the worst case, terminate at an iterate with a small gradient norm but undetected negative curvature. Barring more expensive computations, this situation seems difficult to avoid. However such trouble is, we believe, somewhat unlikely to occur because our accuracy strategy imposes the trust-region subproblem to be solved more accurately when gradients are small (see condition (5.1)) which in turn causes a more thorough exploration of the space, itself increasing the probability of detecting negative curvature, if any.

## 5.4 Performance and comparisons

In the remainder of this chapter, several algorithmic variants are considered and compared on the set of CUTEr test problems. We also present a numerical comparison of FILTRUNC with another widely used package for large-scale optimization, namely LANCELOT-B. A brief description of this software is given in Section 5.4.3. These various comparisons are, in particular, based on number of iterations, conjugate-gradient iterations and total CPU time.

The numerical results of this section will be presented using the performance profiles described in Section 5.2.

### 5.4.1 Filter versus pure trust-region variants

Our first concern is to see if the use of a multidimensional filter technique combined with a trust-region scheme provides benefit compared to a more classical trust-region approach. We then start by analysing the performance of our algorithm by comparing the *filter* variant of FILTRUNC with the *pure trust-region* one, both described in Section 5.3.

Our first observation is that the filter variant is just as reliable as the pure trust-region one. Indeed, on the 159 problems, both the filter and the pure trust-region versions successfully solve 143 problems (within the prescribed iteration and CPU limitations).
For problems where both variants succeed, they report the same final objective function value. Appendix A presents, for the filter and the pure trust-region variant of FILTRUNC, the details of the runs for problems listed in Table 5.3, in particular the number of iterations and conjugate-gradient iterations, the CPU time and the final objective function value. Failure occurs because the maximal iteration count is reached before convergence is declared, except for problems `ARGLINB` and `ARGLINC` that are judged to be too ill-conditioned by both variants, and for problem `MEYER3` where the pure trust-region variant stops for the same reason. Both variants

fail on CHAINWOO, HYDC20LS, LMINSURF, LOGHAIRY, MEYER3, NLMSURF, NONCVXU2, NONCVXUN, SBRYBND, SCOSINE and SCURLY10 because the maximum number of iterations has been reached. Furthermore, if we closely analyse the results, it can be seen that, for some problems, the solution given by the pure trust-region variant is very close to the problem solution although it reports that the step is too short to allow for further progress. This is often caused by the ill-conditioning of the problems. This situation occurs for problems DJTL, FREUROTH, JENSMP, PALMER1D, PENALTY2 and PENALTY3. This is well-known that, for instance, PENALTY3 is ill-conditioned. These failures could possibly be avoided by using preconditioning. But note that we have counted these occurrences as successful in the discussion of this chapter. The filter variant also fails on the three following problems FMINSRF2, FMINSURF and MINSURF, while the pure trust-region one breaks down on MARATOSB, RAYBENDL and RAYBENDS because they have reached the maximum number of iterations.

Figures 5.1, 5.2, and 5.3 give the performance profiles for both variants for number of iterations, CPU time, and the total number of conjugate-gradient iterations, respectively.

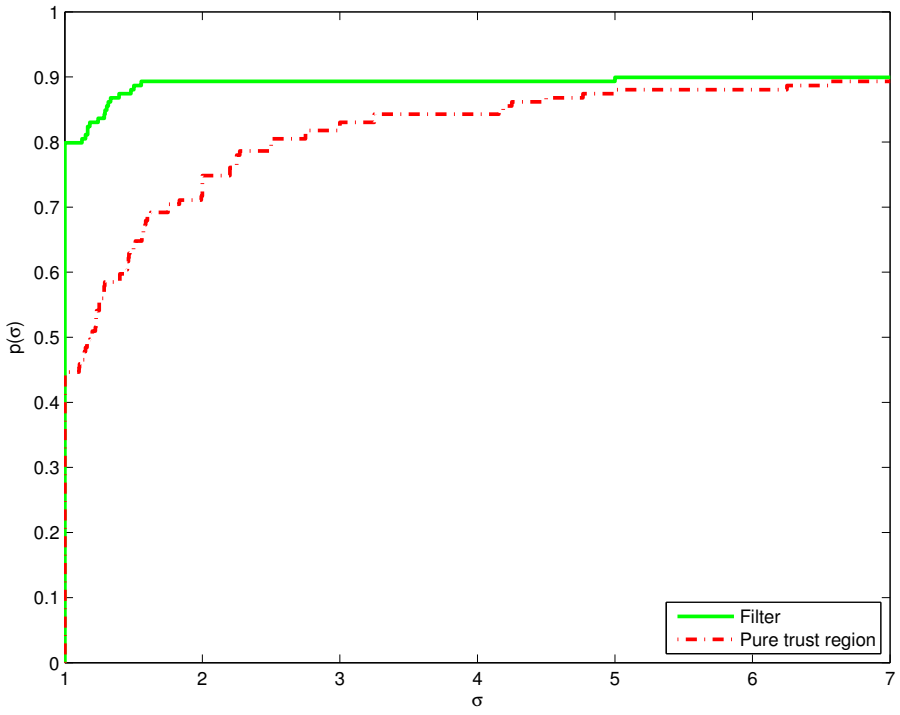


Figure 5.1: Iteration performance profile

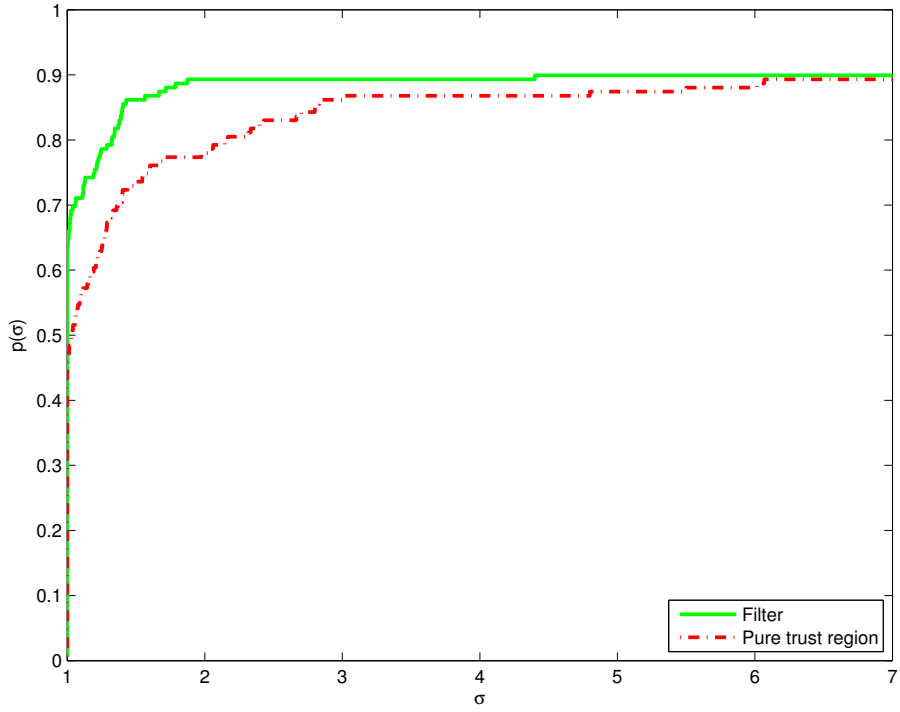


Figure 5.2: CPU performance profile

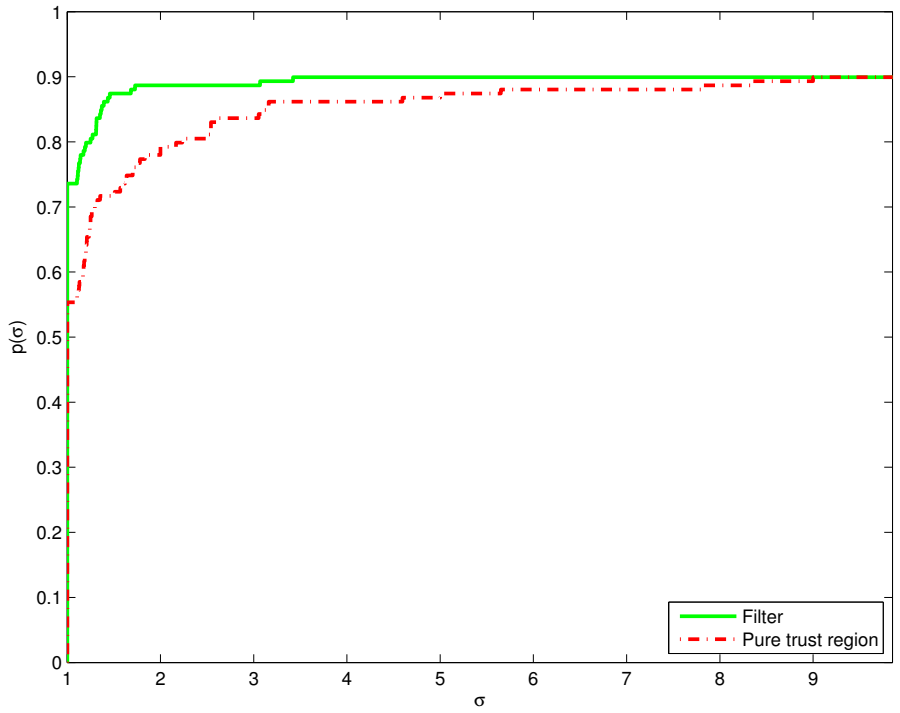


Figure 5.3: CG iteration performance profile

In Figure 5.1, it can be seen that the filter variant is the best solver, in terms of number of iterations, on 80% of the problems while the pure trust-region one is the best in more or less 45% of the cases. Figure 5.2 shows that the filter variant is the faster on approximately 66% of the test problems.

Note that, for simplicity of presentation, we have limited the ratio of performance to a maximum value on the graphs (generally smaller or equal to ten). The right-hand part of the performance curves is therefore not always directly related to overall reliability of the variants; this is why we have mentioned above the exact number of problems solved by both variants.
So the filter variant is significantly more efficient than the pure trust-region one in terms of number of iterations (which is identical to the number of function evaluations minus one). While Figures 5.2 and 5.3 show that its advantage is smaller though still significant in terms of CPU time and conjugate-gradient iterations.

Interestingly, the cost of managing the filter does not appear to dominate the calculation, despite the potentially large number of entries. A closer look at the results shows that the number of filter entries is distributed as indicated in Table 5.4.

| # of filter entries | # of problems |
|---|---|
| $0 \leq n_f \leq 5$ | 117 |
| $6 \leq n_f \leq 10$ | 11 |
| $11 \leq n_f \leq 50$ | 11 |
| $n_f > 50$ | 4 |

Table 5.4: The distribution of the number of filter entries

We can see that the number of filter entries exceeds 50 for 4 problems only: `EIGENBLS` (85 entries), `RAYBENDS` (103 entries), `SCURLY20` (217 entries), and `SCURLY30` (221 entries). Note that the problem `RAYBENDS` could not be solved by the pure trust-region method. Moreover, we did not observe any obvious correlation between filter size and number of variables. Indeed, the largest problem `CLPLATEA`, which has 10100 variables, needs only 7 filter entries and, for the 9 problems for which the dimension is equal to 10000, the number of filter entries does not exceed 11. Furthermore, problems which require the most filter entries, namely `SCURLY20` and `SCURLY30`, have both only 100 variables. Finally, we want to point out that filter resets do not occur very often, so the number of filter entries is not small because the filter is often emptied. If we closely analyse the results, we can observe that the number of times a non-empty filter is reset is zero for more than 70% of the problems, is 1 for more or less 15%

and is smaller than 4 for 5 problems. The number of resets is large for very few problems. For example, 61 resets have occurred for problem GENROSE.

### 5.4.2 Comparison on quadratic programs

Here we compare the performance of the filter and the pure trust-region variants on quadratic programming problems. This time, we have extracted from the CUTEr collection problems with the following features :

```
Objective function type          :  Q (quadratic)
Constraints type                 :  U (no constraint), X (fixed variables only)
Regularity                       :  R (regularity)
Degree of available derivatives  :  2 (analytical second derivatives)
Problem interest                 :  *
Explicit internal variables      :  *
Number of variables              :  *
Number of constraints            :  *
```

This results in the 14 problems given in the following table.

| Problem | $n$ | Problem | $n$ |
|---|---|---|---|
| DIXON3DQ | 10000 | PALMER2C | 8 |
| DQDRTIC | 5000 | PALMER3C | 8 |
| HILBERTA | 2 | PALMER4C | 8 |
| HILBERTB | 10 | TESTQUAD | 5000 |
| MARATOSB | 2 | TOINTQOR | 50 |
| PALMER1C | 8 | TRIDIA | 5000 |
| PALMER1D | 7 | ZANGWIL2 | 2 |

Table 5.5: The quadratic problems and their dimension

Figures 5.4, 5.5 and 5.6 illustrate, as it can be expected, that the filter variant performs very well on this set of unconstrained quadratic problems in terms of number of iterations, CPU time and conjugate-gradient iterations, respectively[(6)].

This indicates that the underlying method works very well on quadratic programs without the restriction to the trust region. So, for quadratic problems, the use of unrestricted steps is

[(6)] In Figures 5.4, 5.5 and 5.6, the curve corresponding to the filter variant coincides with the vertical axis.

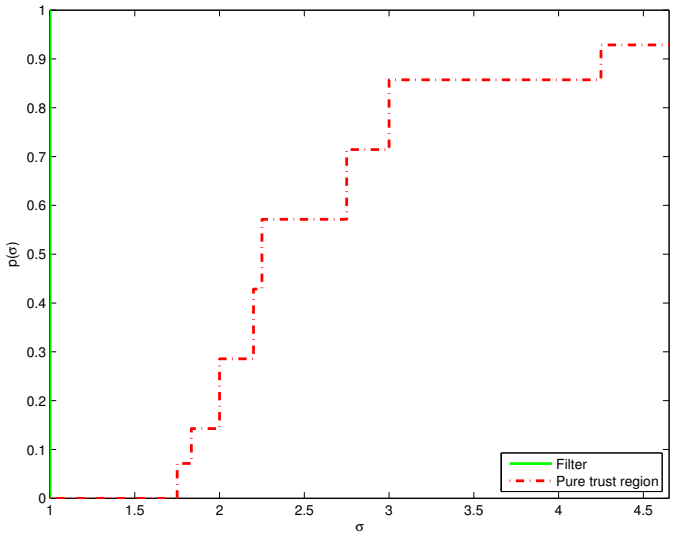


Figure 5.4: Iteration performance profile for quadratic problems

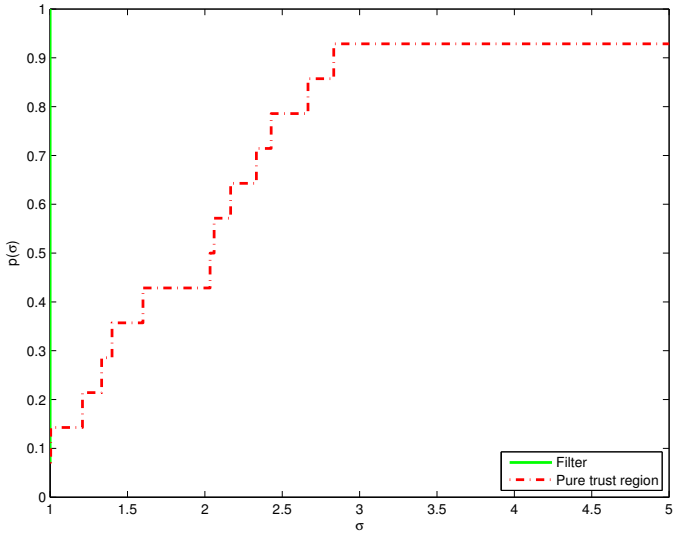


Figure 5.5: CPU performance profile for quadratic problems

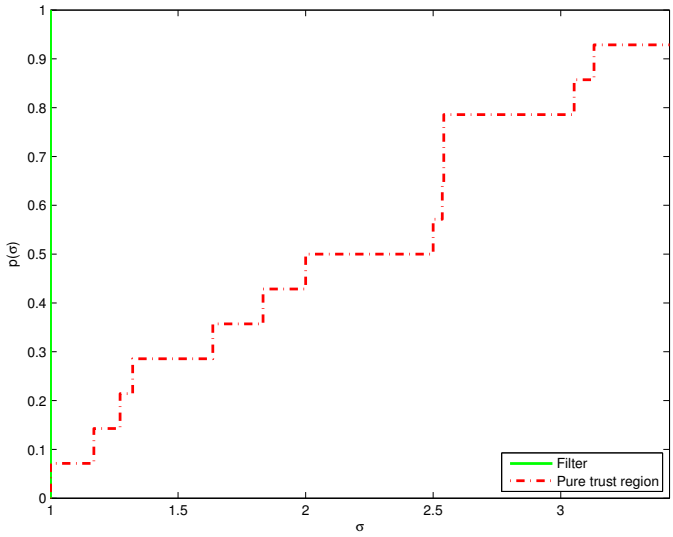


Figure 5.6: CG iteration performance profile for quadratic problems

crucial. The three figures show that the filter variant is the best solver on 100% of the quadratic problems for the three considered criteria. We can observe in Figure 5.7 the evolution of the objective function value for the quadratic problem `DQDRTIC`. We clearly see that the pure trust-region method is slowed down by the trust-region restriction on the step.

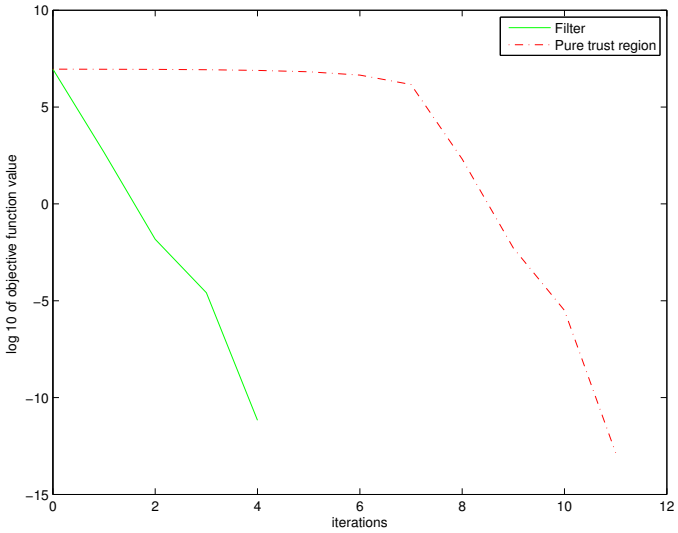


Figure 5.7: The objective function value as a function of the iteration progress on the `DQDRTIC` problem. The filter variant converges first followed by the pure trust-region one.

Obviously, if we performed the pure trust-region variant with a very large initial trust-region

radius, the pure trust-region variant would be more competitive with the filter variant.

### 5.4.3 Comparison with LANCELOT-B

We now include a comparison with LANCELOT-B, one of the codes available in the GALAHAD library [70].

Before presenting the results, we give a brief description of LANCELOT, which is a Fortran package for solving large-scale nonlinearly constrained optimization problems and which has been developed by Conn, Gould and Toint [33]. This algorithm combines the objective function and constraints more complicated than simple bounds on the variables in an augmented Lagrangian function as explained in Section 2.4.2. Then it solves a sequence of subproblems by approximately minimizing the current augmented Lagrangian function within the feasible region defined by the simple bounds. The algorithm used to solve this bounded problem combines a trust region, a gradient-projection method and special data structures to exploit the group partially separable structure of the problem under study. Among other algorithmic options, the LANCELOT package proposes direct or iterative linear solvers for solving the subproblem, analytic or finite-difference derivatives, a lot of preconditioning and scaling techniques, Quasi-Newton methods, and so on.

We found interesting to compare our numerical results with those obtained with LANCELOT-B [68], an updated version of LANCELOT, because this latter allows a non-monotone behaviour in the trust-region algorithm (see Toint [121] or Conn et al. [34, Section 10.1]). We have used this solver without preconditioning, with the initial trust-region radius $\Delta_0$ equals to one and with exact first and second derivatives. The trust region was defined by the Euclidean norm as in our algorithm. The solver LANCELOT-B was processed with its other settings at their default values.

Figures 5.8, 5.9, and 5.10 give the performance profiles for both variants and LANCELOT-B for iteration count, CPU time, and the total number of conjugate-gradient iterations, respectively.

LANCELOT-B, which successfully solves 140 out of 159 problems, is marginally less robust than the other two variants (both solve 143 problems). It does not solve the same list of problems common to both variants and also fails[(7)] on `FMINSRF2` like the filter variant,

[(7)] Failures occur because the maximal iteration count has been reached.

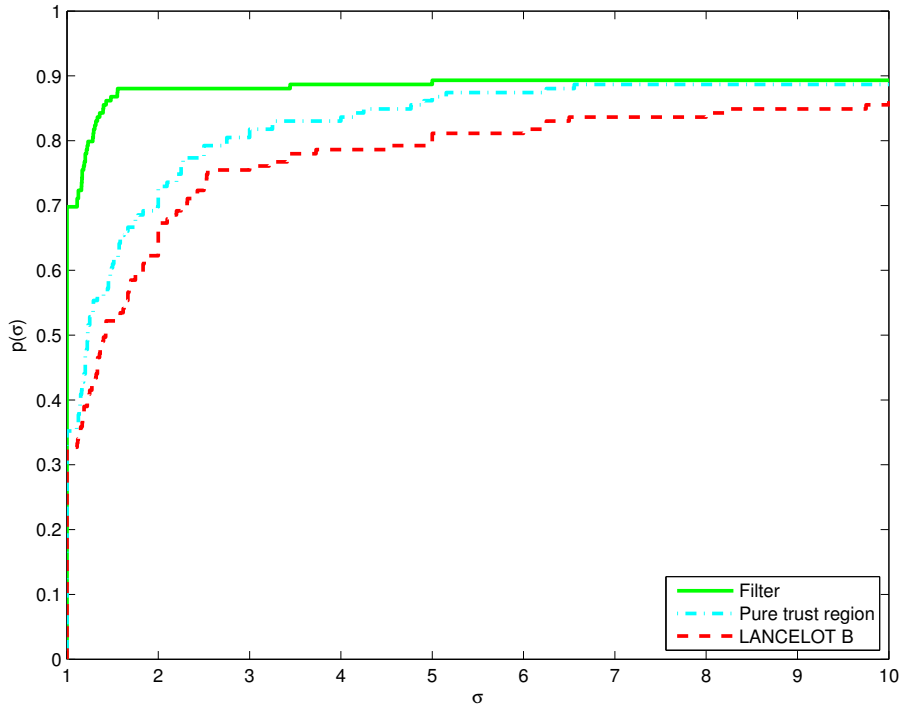


Figure 5.8: Iteration performance profile for both variants and LANCELOT-B

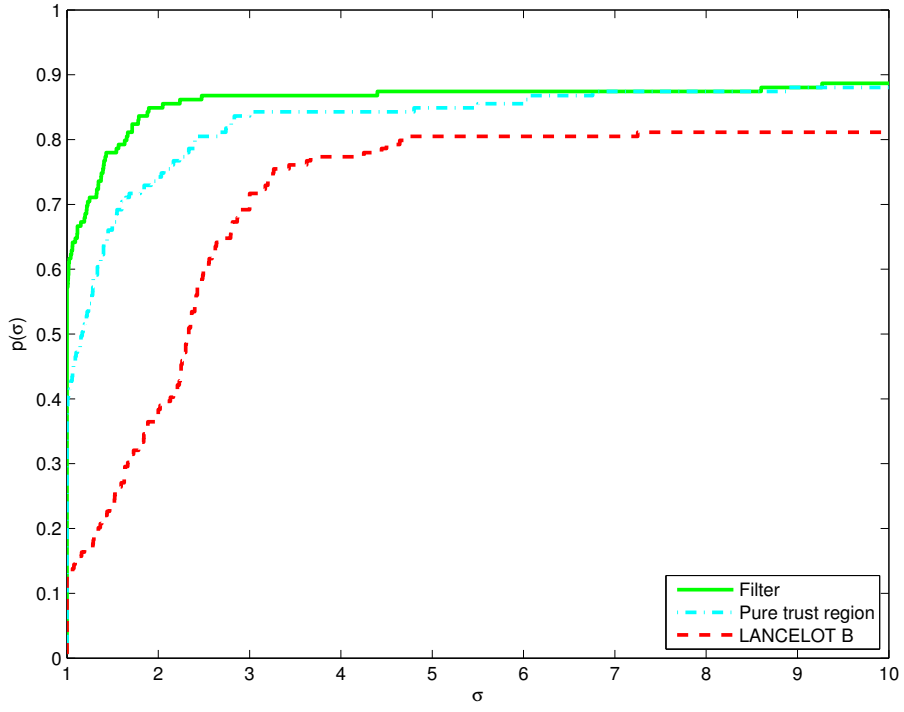


Figure 5.9: CPU performance profile for both variants and LANCELOT-B

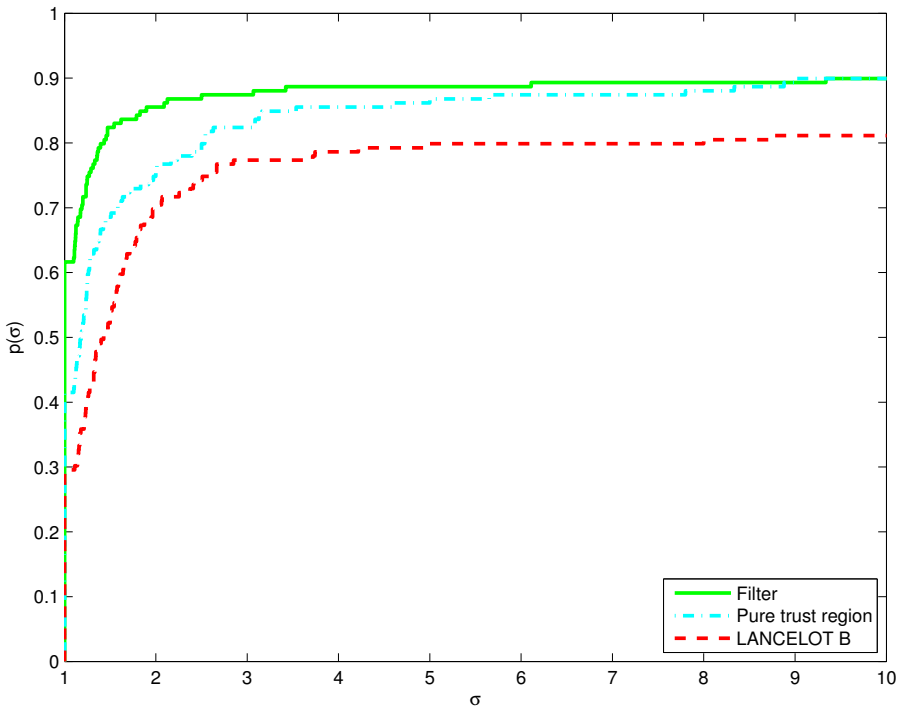


Figure 5.10: CG iteration performance profile for both variants and LANCELOT-B

on RAYBENDL and RAYBENDS like the pure trust-region one and finally on SCURLY20, SCURLY30 and VIBRBEAM. In view of the graphs depicted in Figures 5.8, 5.9, and 5.10, this solver appears to be consistently inferior to the new filter algorithm and also to our pure trust-region variant. So, in our tests, the filter variant is clearly the winner compared to LANCELOT-B, both in reliability and efficiency.

This comparison is interesting in that it suggests not only that the improved performance of the new algorithm might be due to the non-monotone nature of the mechanism to accept new iterates, but also that the capability to use steps that extend beyond the trust-region boundaries is crucial (see also Section 5.4.5).

We next present in Figure 5.11, 5.12 and 5.13 some plots of the evolution of the objective function value for the filter and pure trust-region variants, as well as for LANCELOT-B. These plots are typical of the cases where the new algorithm outperforms the others. For this algorithm, we note, especially in Figure 5.11, the large oscillations in objective value prior to convergence. Looking at these figures, it is remarkable that the algorithm is nevertheless provably convergent.

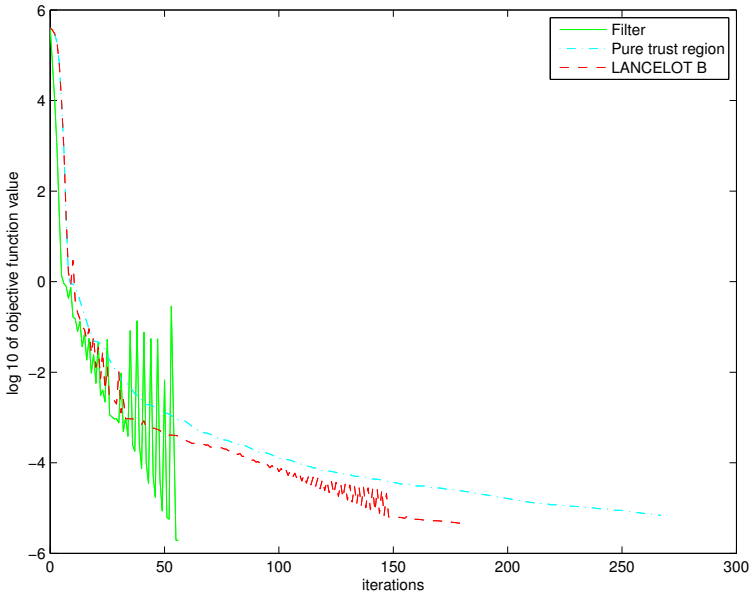


Figure 5.11: The objective function value as a function of the iteration progress on the EXTROSNB problem for both variants and LANCELOT-B. The filter variant oscillates the most and converges first, followed by the moderately non-monotone LANCELOT-B, itself followed by the monotone pure trust-region variant.

It can also been remarked in Figure 5.11 that, for the LANCELOT-B solver, there is a notion similar to *"decrease in the long run"* that we do not have with the filter variant.

However, this behaviour is not observed for all problems and sometimes the filter variant, although there are large oscillations in objective function value, is the last to converge. This happens, for example, for the problem TOINTPSP for which the evolution of the objective function value is represented in Figure 5.14. For this problem, the filter variant has added 40 points in the filter.

The last plots of this section represent the evolution of the gradient norm for problem EXTROSNB. For the sake of clarity, we have divided the presentation into two pictures. Figure 5.15 gives the evolution of the gradient norm for the filter and the pure trust-region variants, while Figure 5.16 is the comparison of the filter variant against LANCELOT-B. Obviously, the behaviour in terms of gradient norm is non-monotone for the three solvers, but again, we can observe the larger oscillations in gradient norm value described by the filter variant.

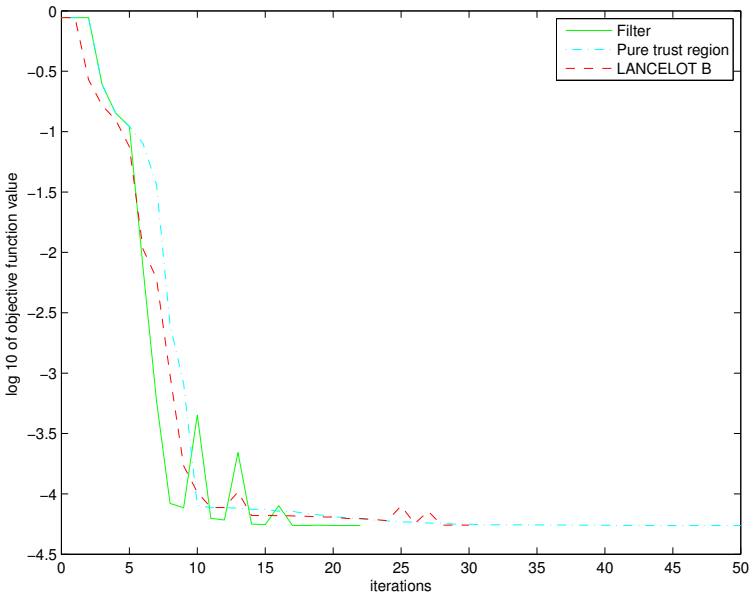


Figure 5.12: The objective function value as a function of the iteration progress on the OSBORNEA problem for both variants and LANCELOT-B

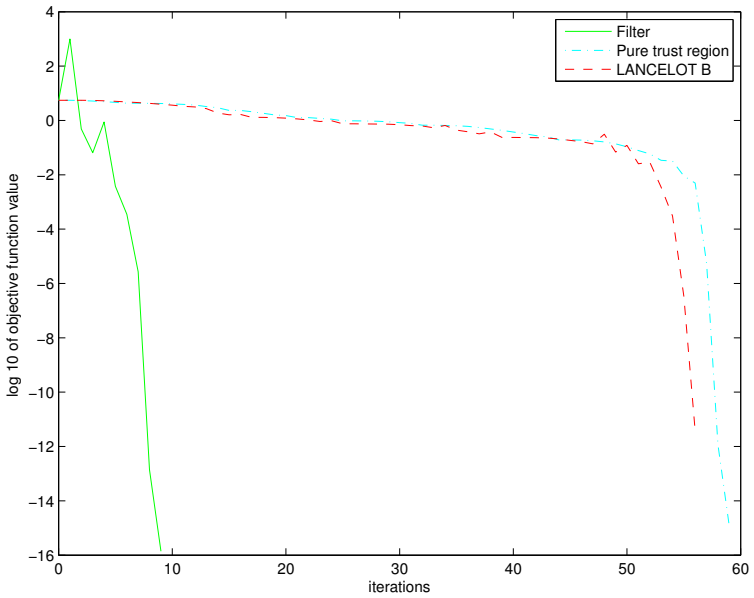


Figure 5.13: The objective function value as a function of the iteration progress on the SINEVAL problem for both variants and LANCELOT-B

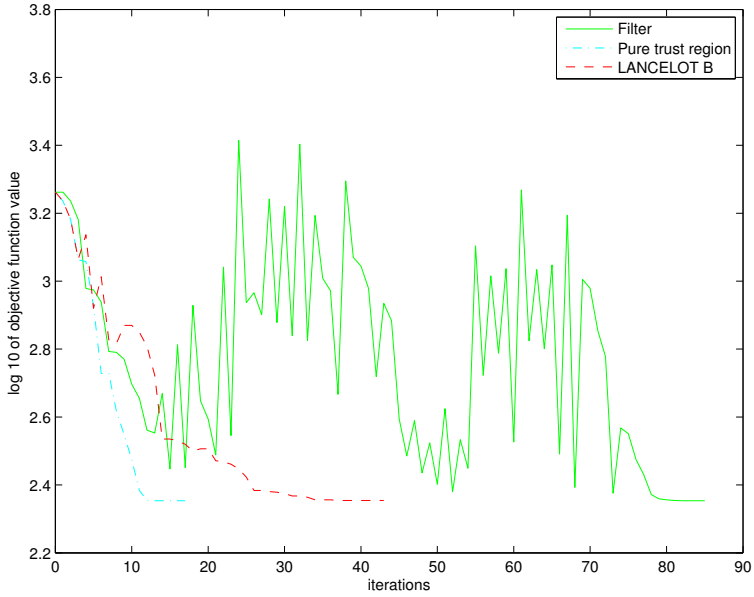


Figure 5.14: The objective function value as a function of the iteration progress on the TOINTPSP problem for both variants and LANCELOT-B

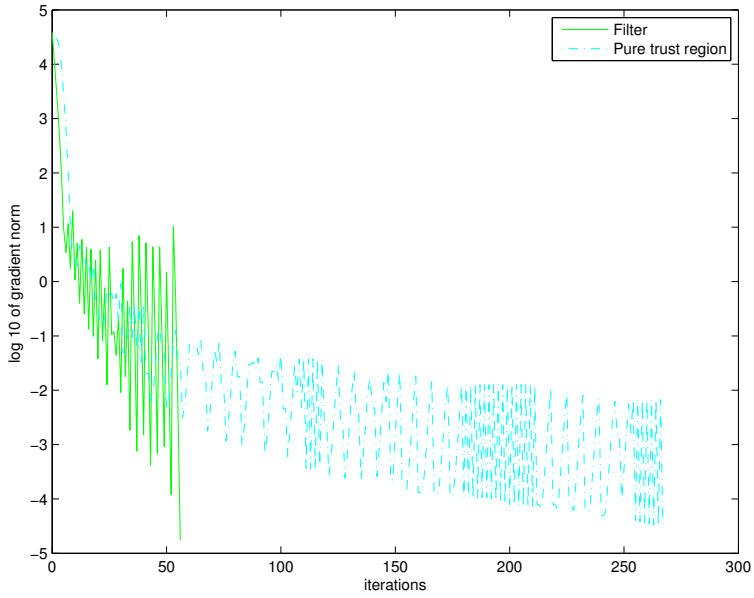


Figure 5.15: The gradient norm as a function of the iteration progress on the EXTROSNB problem for both variants

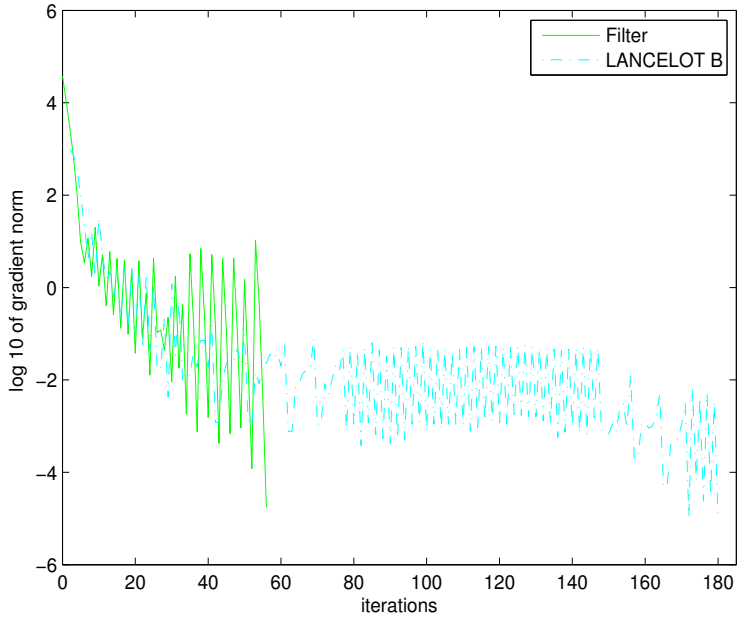


Figure 5.16: The gradient norm as a function of the iteration progress on the EXTROSNB problem for the filter variant and LANCELOT-B

### 5.4.4 Algorithmic variants

In what follows, we will compare the filter variant of FILTRUNC with two other algorithmic variants for reliability and efficiency.

We first consider what happens if we limit the number of filter entries to 50, this is the variant we call *limited*. We have already discussed in Section 5.4.1 some statistics about the maximum number of filter entries observed in our numerical results. These observations suggest that the filter size typically remains very modest. Furthermore, the cost of the acceptance tests associated with filter entries is typically not too high; indeed it is enough to find a single decreasing gradient component to decide acceptability of a trial point with respect to a filter entry. Moreover, keeping track of the Euclidean norm of the various filter entries helps in avoiding unnecessary comparisons. The details of these strategies, known as *pre-filtering*, are discussed in Gould and Toint [76].

Nevertheless, at least in theory, nothing prevents the filter size from growing, possibly to infinity. Practically, a very large number of points might therefore be required, although this does not happen in our tests, and this could, again in principle, be a serious drawback, especially for large-scale problems where each filter point has itself a large number of components. For-

tunately, this problem can be fixed without sacrificing our convergence guarantee. Should the problem arise in that, at some iteration, the total storage for filter points reaches a user-defined upper limit, two different techniques can be used to continue the calculation. The first idea is simply to revert to a pure trust-region scheme from that iteration on. Admittedly, we would then loose some of the potential benefits of using a filter technique, but convergence is not put at risk. The second strategy is a progressive form of the first. As indicated in the paper of filter methods for nonlinear equations of Gould et al. [65], the components of the gradient can be grouped in progressively larger sets and the filter entries are now defined as the Euclidean norm of the sub-vector of components belonging to the set. This results in a progressive decrease of the amount of storage required to store the entire filter. In the limit where a single component set is considered and assuming dominated filter points are removed, the filter reduces to a single number, *i.e.* an upper bound on the Euclidean norm of the gradient, thus eliminating all storage problems.

Here we have considered to limit the number of filter entries to 50 and to go back to a classical trust-region method if this number is reached. It can be seen in Figures 5.17, 5.18 and 5.19 a slight improvement obtained by the limited variant. This is mostly due to problems `FMINSURF`, `FMINSURF2` and `MINSURF` for which this variant ensures convergence in less than 150 iterations while the default filter variant fails to converge in 1000 iterations. Note that the unlimited filter variant has added more than 400 points to the filter for these three problems.

We also discuss the effect on our numerical results of considering *signed filter entries* (which means that we consider the absolute values of filter entries) instead of *unsigned* ones as described in Section 5.3. So we now use the filter test acceptance mechanism (4.9). This algorithmic variant is referred as the *signed* variant.

The relative efficiency and robustness of these two algorithmic variants compared to the default filter variant are illustrated by the performance profiles of Figures 5.17, 5.18 and 5.19. Our numerical experiments show a slightly increased reliability for the limited variant, in fact this latter successfully solved 146 problems out of 159 while the default filter variant solved 143. We also obtain a modest gain in efficiency, in terms of iterations, CPU time and conjugate-gradient iterations. On the other hand, the signed variant appears to be a little less reliable (142/159 versus 143/159) and also less efficient. Signed and unsigned variants successfully solved the same problems except that the signed variant fails to solve the problem `SCURLY30` because the maximum number of iterations is reached before convergence is declared. But note that the unsigned variant solves this problem in 906 iterations. The benefit achieved by the unsigned default variant compared to the signed variant is obtained at the cost of including more entries in the filter, as shown in Figure 5.20.

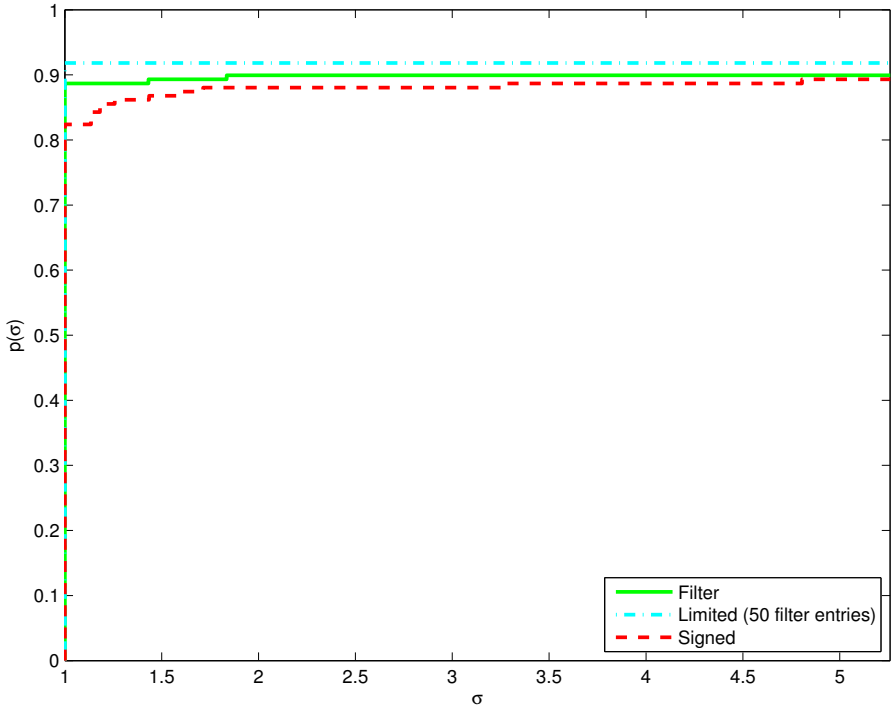


Figure 5.17: Iteration performance profile for the three filter variants

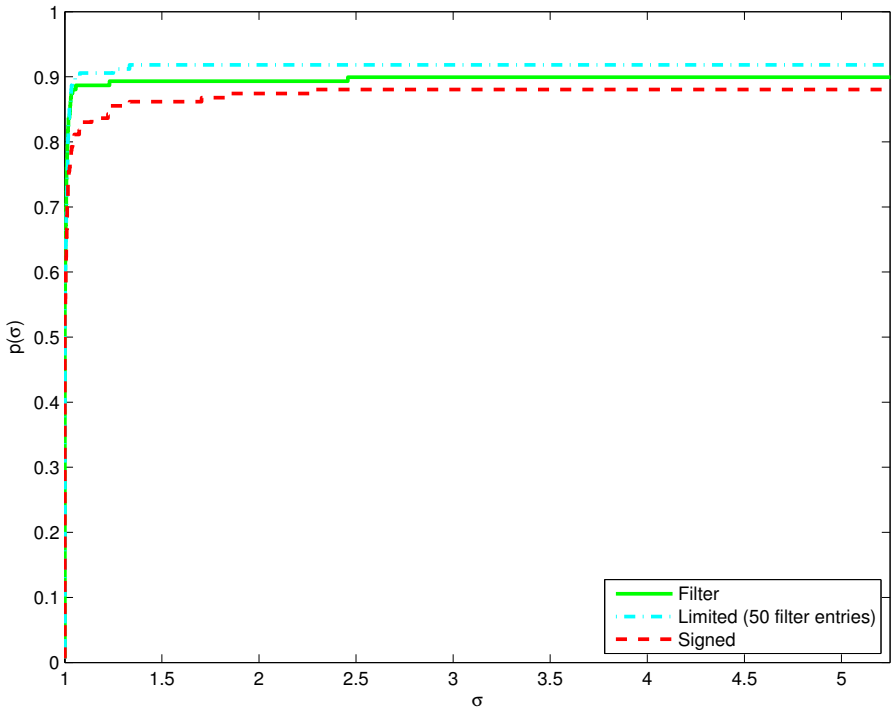


Figure 5.18: CPU performance profile for the three filter variants

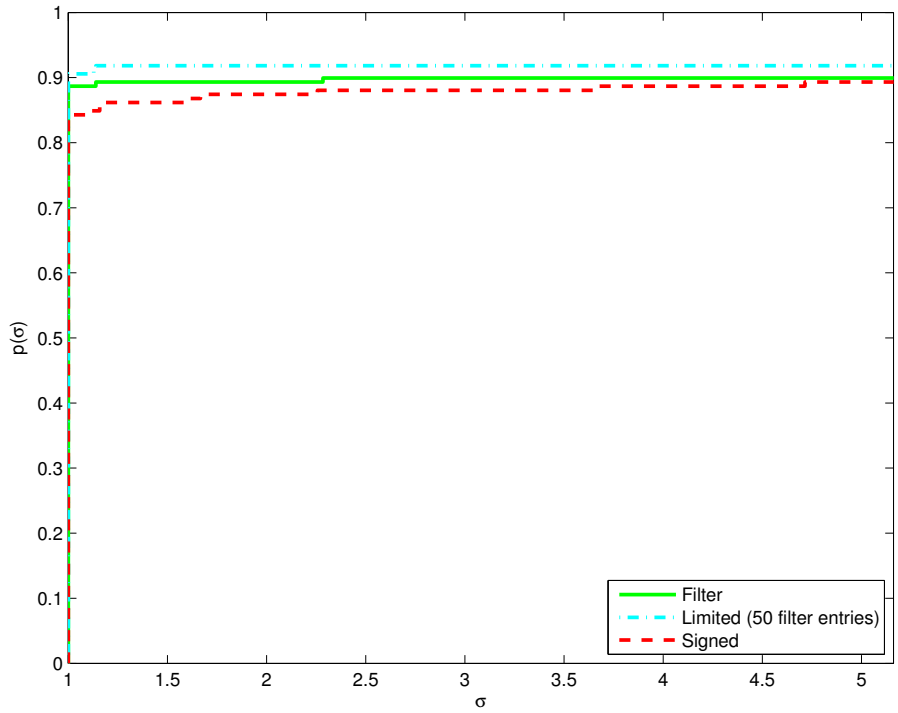


Figure 5.19: CG iteration performance profile for the three filter variants

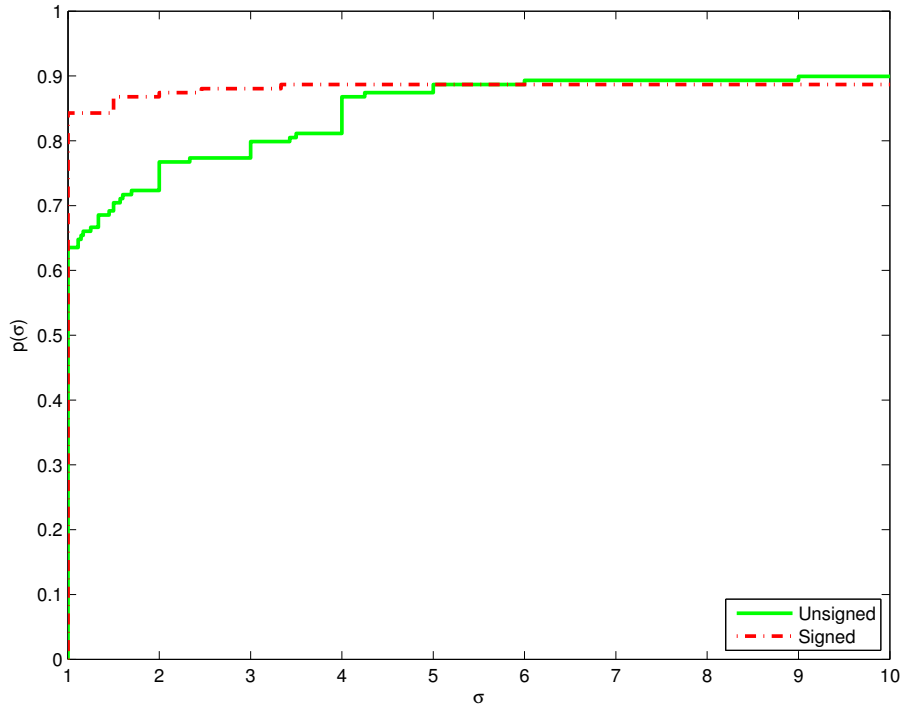


Figure 5.20: Filter size performance profile for the filter variants using unsigned and signed filter entries

### 5.4.5 Unrestricted steps

In this last section, we will compare the filter and the pure trust-region variants with a version of our code without the filter mechanism but with the possibility of taking unrestricted steps. This comparison will permit to see if the use of unrestricted steps that extend beyond the trust-region boundary is an important feature of our filter algorithm. Figures 5.21, 5.22 and 5.23 give the performance profiles for number of iterations, CPU time, and the total number of conjugate-gradient iterations, respectively.

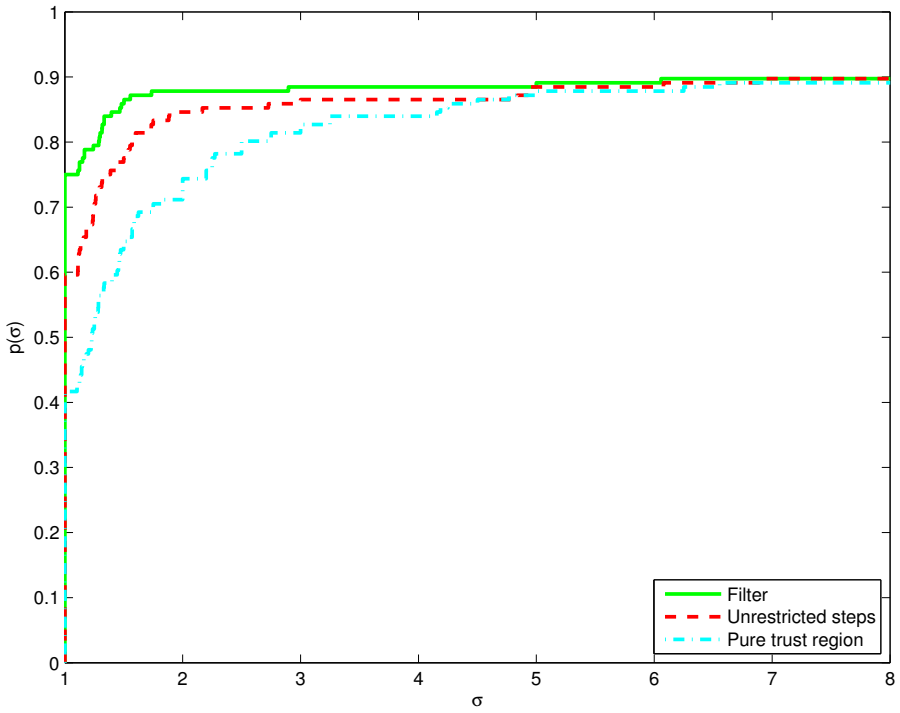


Figure 5.21: Iteration performance profile

We can see that the fact of allowing unrestricted steps improves the performance of the traditional trust-region method (especially in terms of number of iterations). However, we can observe that our filter method remains the best variant for all considered performance measures. This indicates that this is the combination of the multidimensional filter with the use of unrestricted steps which allows to obtain the good performance of our method.

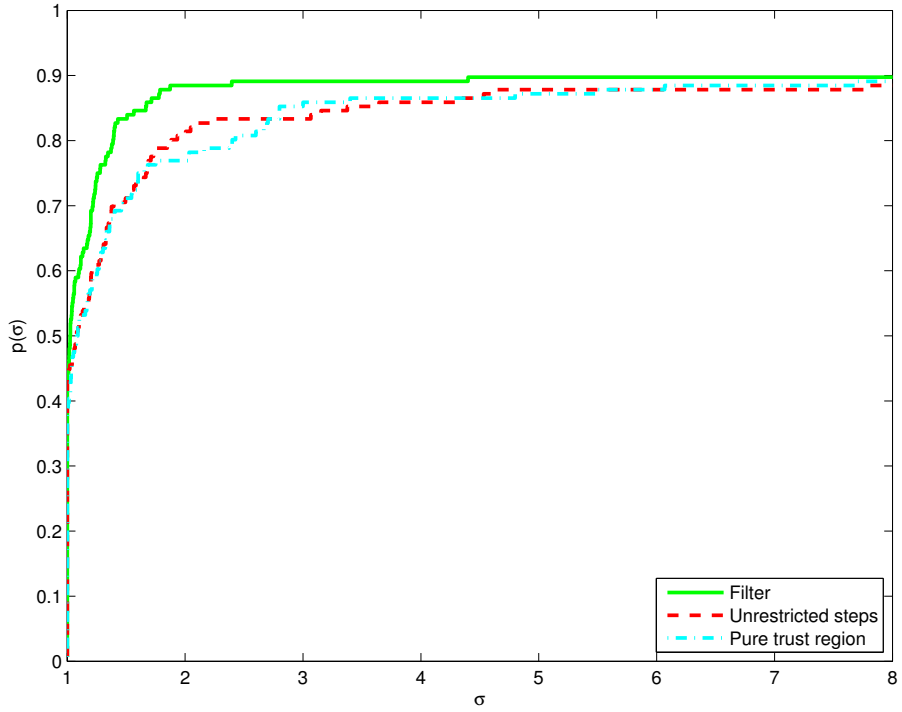


Figure 5.22: CPU performance profile

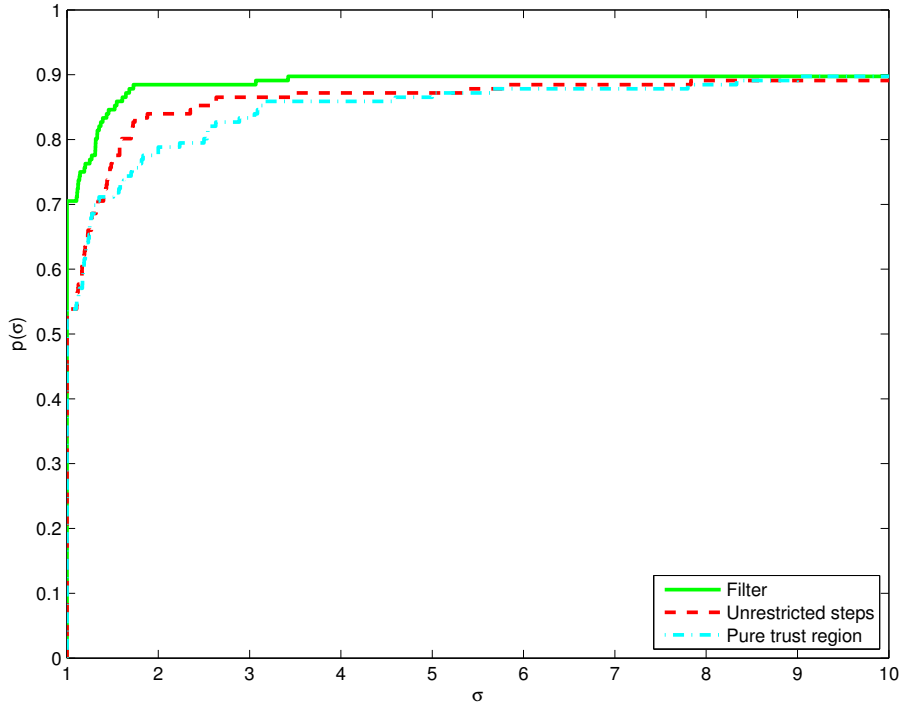


Figure 5.23: CG iteration performance profile

## 5.5 Conclusion

In conclusion, our numerical experience on the set of unconstrained test problems from the CUTEr collection indicates that the efficiency gains introduced by the combination of the multidimensional filter technique and a trust-region method appear to be significant in terms of number of iterations, CPU time and number of conjugate-gradient iterations. Finally, we emphasize the fact that the use of unrestricted steps is a filter technique's crucial feature, and that only allowing a non-monotone behaviour in the objective function value is not likely to be as successful. This is clearly shown when comparing the results of the filter variant to those of LANCELOT-B, which is itself a non-monotone trust-region method but with restricted steps.

# Chapter 6

# Do approximate derivatives hurt filter methods?

In this chapter we will present an experimental study performed to determine the influence of using approximate first and/or second derivatives of the objective function on our filter-trust-region algorithm designed for solving unconstrained nonlinear optimization problems. Our aim is to answer the following question :

> *Does the use of approximate derivatives damage the efficiency and the robustness of methods based on the filter mechanism ?*

Numerical experiments presented below and carried out on small-scale unconstrained problems from the CUTEr collection describe the effect of using approximate derivatives on the performance of Algorithm 4.1; this study is also presented in Sainvitu [110].

## 6.1 The framework

As most algorithms for nonlinear optimization, the filter-trust-region algorithm proposed in Chapter 4 for solving the following unconstrained minimization problem

$$\min_{x \in \mathbb{R}^n} f(x) \tag{6.1}$$

requires knowledge of first and second derivatives of $f$. In the implementation used to perform our numerical experiments in Chapter 5, the derivatives need to be calculated analytically and supplied by the user. However, in some situations, the first and *a fortiori* the second derivatives of the function involved in problem (6.1) may be unavailable. This may be due to the fact that the evaluation of these derivatives is very difficult, time-consuming or their calculation requires

the solution of another problem. Notwithstanding, the unavailability of the derivatives does not necessarily imply that the objective function we consider is not differentiable. Actually, in the remainder of this chapter, we assume that the objective function $f$ is indeed twice continuously differentiable.

One way to fix the issue of unavailability of derivatives is to use finite-difference approximations to the gradient and the Hessian matrix (see Section 1.3.4). Another one, which is more widespread, is to use secant approximations to the second derivative matrices by applying, for instance, BFGS or SR1 updates (see Section 2.3.1). It is also possible, as mentioned in Section 1.3.5, to treat problems with unavailable derivatives by way of automatic differentiation techniques. But we have decided not to consider this approach in our numerical analysis. We have also explored the influence of a perturbation of second-order derivatives on the efficiency and the robustness of our algorithm.

Our first step in this investigation is to see where the first and second derivatives appear in the filter-trust-region algorithm. They are both present in different positions of Algorithm 4.1. The gradient and the Hessian are obviously taken into account in the definition of the model of the objective function and are thus important for the computation of the next trial point.
We can guess that the approximation to the gradient will be a very critical issue because it is precisely the quantity we are trying to make zero. So, as in classical unconstrained minimization algorithms, the gradient of the objective function must be known accurately both for computing the next trial point and for the stopping criterion. In our algorithm, the first derivative has also a major role because it determines the filter. Indeed, the gradient appears in the definition of the filter and thus in the filter test acceptance mechanism (4.9).
The second derivatives notably influence the fact that the algorithm chooses to use the filter technique or not. When a negative curvature is detected, we just fall back to a classical trust-region method and the step is restricted to the trust region. Therefore, if the second-order information is not exact and the algorithm detects erroneous negative curvature, the convergence may be slowed down by smaller steps.

Before examining the impact of using approximate derivatives, we briefly discuss the convergence of methods where the model is made up with estimate first and/or second derivatives. For inexact gradients, we have to require that the absolute error on the objective gradient must go to zero when the gradient of the model itself converges to zero. This condition was proposed and analysed by Carter [20] in the context of trust-region methods using inexact gradient information. For Hessian approximations, we have to impose bounds on the growth of the norm

of these approximations as the iterations proceed. We refer the reader to the books of Conn, Gould and Toint [34, Section 8.4] and Dennis and Schnabel [41, Chapter 9] and to the article of Dennis and Moré [40] for more details about convergence analysis of such methods. Local convergence properties of Quasi-Newton methods are studied in Broyden, Dennis and Moré [16]. Convergence of the Hessian approximation matrix to the exact Hessian of the objective function has notably been studied by Conn, Gould and Toint [30] and Byrd, Khalfan and Schnabel [18] for the SR1 update. Interesting discussions on finite-difference approximations can be found in the book of Dennis and Schnabel [41, Section 5.6] or in the paper of Gill et al. [61]. Boggs and Dennis [11] have analysed the error inherent in using finite-difference gradients in minimization algorithms.

## 6.2 Numerical investigation

As mentioned above, the exact first and second derivatives are used in our first implementation of the filter-trust-region algorithm. So the question we are interested in here is : *is the behaviour of the filter-trust-region algorithm directly related to the use of exact derivatives ?* The aim of this section is to answer this question by studying the performance of the algorithm over a set of test problems if we do not use exact derivatives. We will consider two kinds of derivatives estimations : firstly, we will examine what happens if we approximate the derivatives by finite-difference techniques, and secondly, if we estimate them by secant approximations. The numerical results are again examined by means of performance profiles proposed by Dolan and Moré in [43] and briefly presented in Section 5.2. In this framework, the number of iterations and the total CPU time are naturally the criteria we have chosen for comparing the performances.
It occurs that some variants are very close to the problem solution despite them declaring that no further progress seems possible. As in Chapter 5, we have flagged these occurrences as successful in the discussion of this section. And again we have limited the ratio of performance to some maximum value on the graphs and therefore the right-hand part of the performance profiles does not always directly represent the overall reliability of the different variants.

We now discuss the framework in which our numerical experiments are performed. Obviously, several of our choices are not the only ones possible or even the only ones used and our numerical investigation is thus not perfect.
Our results are obtained by running our algorithm on 66 unconstrained small-scale problems from the CUTEr collection. We have selected problems of small dimension[(1)] from the list of

[(1)]The maximum number of variables is 12.

test problems given in Table 5.3. Note that our aim here is only to see if the advantageous behaviour of filter methods described in many references and, in particular, in Chapter 5, is directly dependent on the use of exact derivatives; this is why we have just considered problems of small size. The numerical study presented in this chapter is intended to demonstrate the importance or not of the exact derivatives in our filter-based approach and not to provide a high performance solver in the case where derivatives are unavailable, which is beyond the scope of this dissertation. The names of the problems with their dimension (i.e. the number of free variables) are detailed in Table 6.1.

| Problem | $n$ | Problem | $n$ | Problem | $n$ |
|---|---|---|---|---|---|
| AIRCRFTB | 5 | EXPFIT | 2 | MEYER3 | 3 |
| ALLINITU | 4 | GROWTHLS | 3 | OSBORNEA | 5 |
| BARD | 3 | GULF | 3 | OSBORNEB | 11 |
| BEALE | 2 | HAIRY | 2 | PALMER1C | 8 |
| BIGGS3 | 3 | HATFLDD | 3 | PALMER1D | 7 |
| BIGGS5 | 5 | HATFLDE | 3 | PALMER2C | 8 |
| BIGGS6 | 6 | HEART6LS | 6 | PALMER3C | 8 |
| BOX2 | 2 | HEART8LS | 8 | PALMER4C | 8 |
| BOX3 | 3 | HELIX | 3 | PALMER5C | 6 |
| BRKMCC | 2 | HIELOW | 3 | PALMER6C | 8 |
| BROWNBS | 2 | HILBERTA | 2 | PALMER7C | 8 |
| BROWNDEN | 4 | HILBERTB | 10 | PALMER8C | 8 |
| CLIFF | 2 | HIMMELBB | 2 | ROSENBR | 2 |
| CUBE | 2 | HIMMELBF | 4 | S308 | 2 |
| DENSCHNA | 2 | HIMMELBG | 2 | SINEVAL | 2 |
| DENSCHNB | 2 | HIMMELBH | 2 | SISSER | 2 |
| DENSCHNC | 2 | HUMPS | 2 | SNAIL | 2 |
| DENSCHND | 3 | JENSMP | 2 | STRATEC | 10 |
| DENSCHNE | 3 | KOWOSB | 4 | VIBRBEAM | 8 |
| DENSCHNF | 2 | LOGHAIRY | 2 | WATSON | 12 |
| DJTL | 2 | MARATOSB | 2 | YFITU | 3 |
| ENGVAL2 | 2 | MEXHAT | 2 | ZANGWIL2 | 2 |

Table 6.1: The test problems and their dimension

In each case, the starting point supplied with the problem was used. As for the numerical results of the previous chapter, all tests were performed in double precision on a workstation with a 3.2 GHz Pentium IV biprocessor and 2 Gbytes of memory under Suse Professional 9.0 Linux and the Lahey Fortran compiler (version L6.10a) with default options.

In practice, we stop the algorithm if

$$\|g_k\| \leq 10^{-6}\sqrt{n},$$

where $g_k$ stands for $\nabla_x f(x_k)$ or an approximation to this gradient, and the flag `NONCONVEX` is unset. Again, all attempts to solve the test problems were limited to a maximum of 1000 iterations or 1 hour of CPU time. As in Chapter 5, the variability of CPU times for small times is taken into account by repeatedly solving the same problem until a threshold of ten seconds is exceeded and then taking the average per run. The different parameters defined in Step 0 of Algorithm 4.1 are set to the values given in Section 5.3. The other practical aspects about the implementation detailed in this latter section remains valid here.

We have also considered both algorithmic variants of the previous chapter, namely the *filter* variant and the *pure trust-region* one. Incorporating our two algorithmic variants in the different techniques used to approximate the derivatives, we obtain about 70 tested variants. For completeness, the behaviour of the two variants using the exact first and second derivatives (referred as the *exact* variants) is also shown in the performance profiles.

### 6.2.1 Finite differences

Firstly, we consider the effect of substituting finite-difference approximations for the analytic derivatives on the algorithm. We also discuss the practical choice of stepsizes in computing these approximations.

#### Finite-difference Hessians

We start by analysing the case where exact first derivatives are available. In Section 1.3.4, we have seen two formulae to approximate the Hessian matrices, the forward (1.14) and the central one (1.15). An important issue in the implementation of these formulae is the choice of the finite-difference stepsize $h$. For this selection, we have to make a compromise between large rounding errors, corresponding to small stepsizes, and large approximation errors, generated by large stepsizes. We have tested different choices for the stepsize components $h_j (j = 1, \ldots, n)$ from the literature which are given in Table 6.2.

| | forward | central |
|---|---|---|
| stepsize 1 | $\sqrt{\epsilon_M}$ | $\sqrt[3]{\epsilon_M}$ |
| stepsize 2 | sign $(x_j)\sqrt{\epsilon_M}\max(\|x_j\|, 1)$ | sign $(x_j)\sqrt[3]{\epsilon_M}\max(\|x_j\|, 1)$ |
| stepsize 3 | $\sqrt{\epsilon_M}\max(\|x_j\|, 1)$ | $\sqrt[3]{\epsilon_M}\max(\|x_j\|, 1)$ |
| stepsize 4 | $\sqrt{\epsilon_M}(1 + \|x_j\|)$ | $\sqrt[3]{\epsilon_M}(1 + \|x_j\|)$ |
| stepsize 5 | - | $10^{-4}(1 + \|x_j\|)$ |

Table 6.2: The different stepsizes

On our test problem set, the best results are obtained with the third stepsize for the forward formula and with the first stepsize for the central one; therefore, we present in this dissertation the performance profiles with these choices for the stepsizes. However, the selected stepsize rules are not very significantly better than the others and the performance profiles would not be very different with other stepsizes. Figures 6.1 and 6.2 show the efficiency in terms of number of iterations and CPU time for filter and pure trust-region variants, both with exact second derivatives and forward or central finite-difference approximations to them.

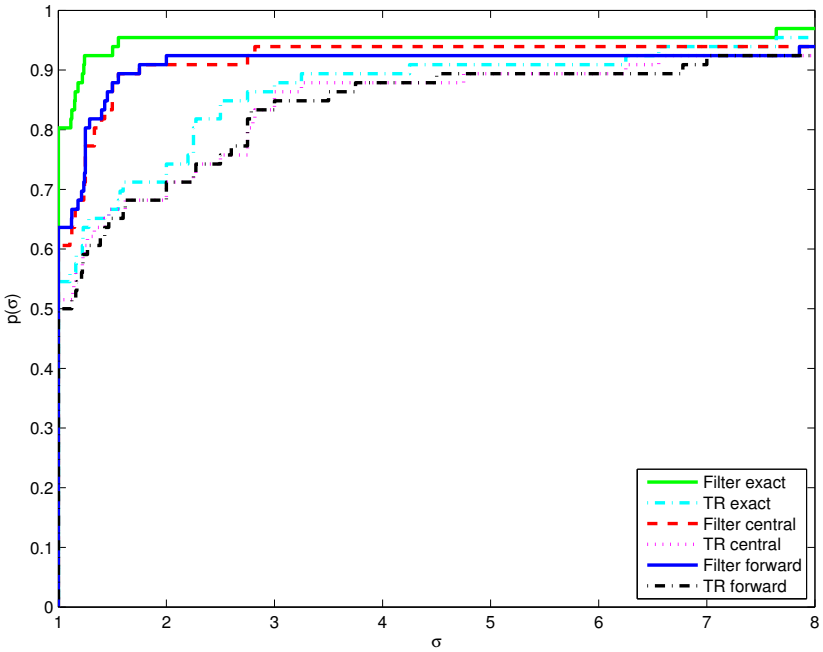


Figure 6.1: Iteration performance profile with exact derivatives and approximate second derivatives by finite differences when the gradient is available

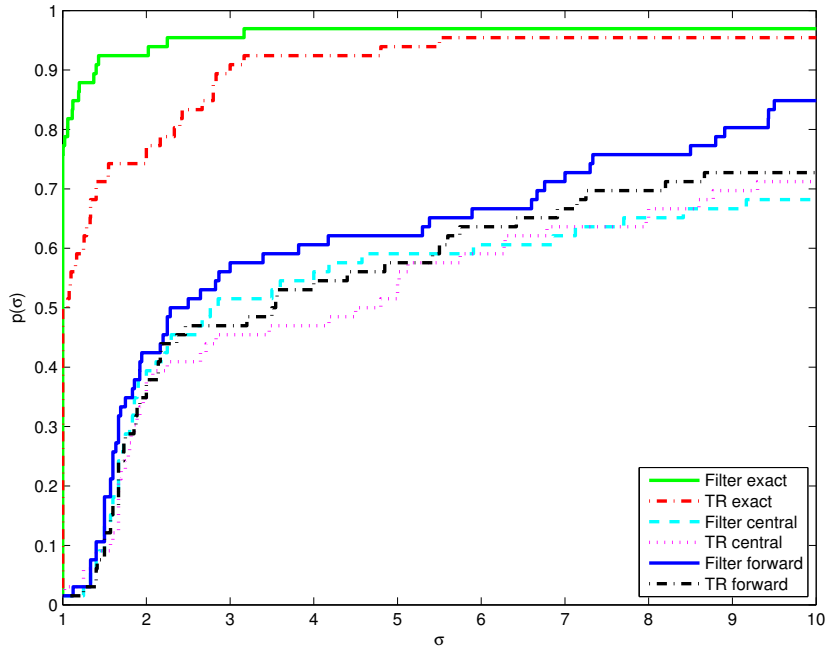


Figure 6.2: CPU time performance profile with exact derivatives and approximate second derivatives by finite differences when the gradient is available

Figure 6.1 indicates that the three filter variants are better than the three corresponding pure trust-region variants if we use the number of iterations as performance criterion. So even the filter versions using approximate Hessian matrices by means of finite differences are more efficient (in terms of number of iterations) than the pure trust-region method with exact derivatives. Both exact variants are obviously the more robust ones.
However, the situation is not the same if we now consider the CPU time for comparing the different variants. It can be observed in Figure 6.2 that the finite-difference variants are, as expected, much more computationally expensive than the exact variants. As we have already mentioned in Section 1.3.4, the finite-difference techniques require additional gradient evaluations. Variants using such approaches are thus much more costly than variants where derivatives are analytically available. Furthermore, approximating the Hessian matrices by the central formula (1.15) requires twice more gradient evaluations than using the forward one (1.14). In Figure 6.2, we can observe that the filter variant with forward finite-difference approximations is the less computationally expensive among all variants with finite differences.

Some small negative curvature in the Hessian matrix may remain undetected when it is approximated by finite-difference formulae. However, we can point out the fact that the fil-

ter algorithmic variant with approximate second derivatives by finite-difference techniques is significantly better as well in terms of iteration count as in terms of CPU time than the corresponding pure trust-region variant. The use of approximate Hessians by finite differences within a filter-trust-region framework does not seem to have more influence than such an approximation in the more classical trust-region scheme. The plots in Figures 6.3 and 6.4, showing the iteration and CPU time performance profiles for the filter and the pure trust-region variants with Hessians approximated by forward finite differences, clearly illustrate this fact.

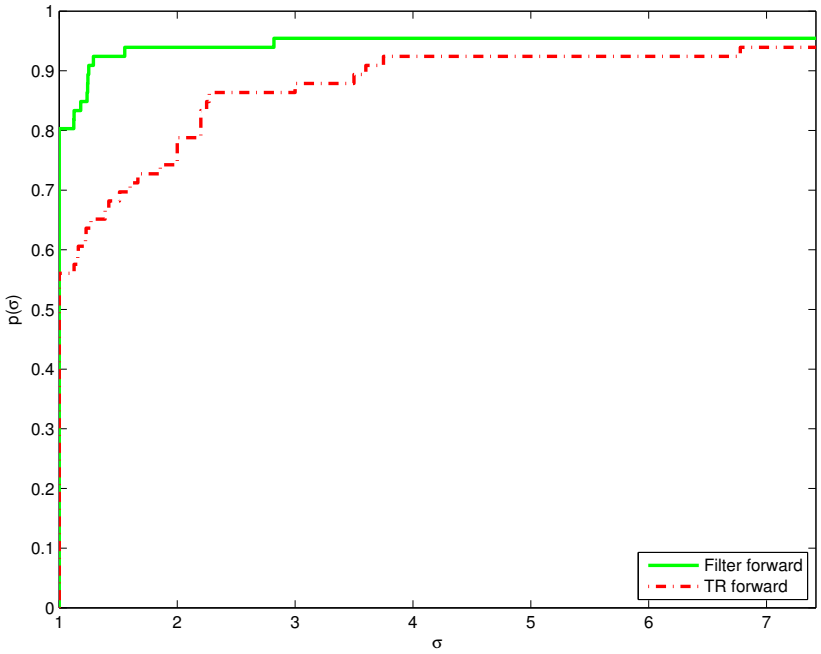


Figure 6.3: Iteration performance profile with approximate second derivatives by forward finite differences when the gradient is available

### Finite-difference gradients and Hessians

We now examine the case where even the gradient of the objective function is not available; so both first and second derivatives are to be approximated by finite-difference formulae. The gradient $\nabla_x f(x)$ is approximated either by the forward formula (1.12) or by the central one (1.13) while the approximation of the Hessian matrix is built with (1.16), which uses only function values.

The tested stepsizes in the case where the first derivative is approximated by a forward for-

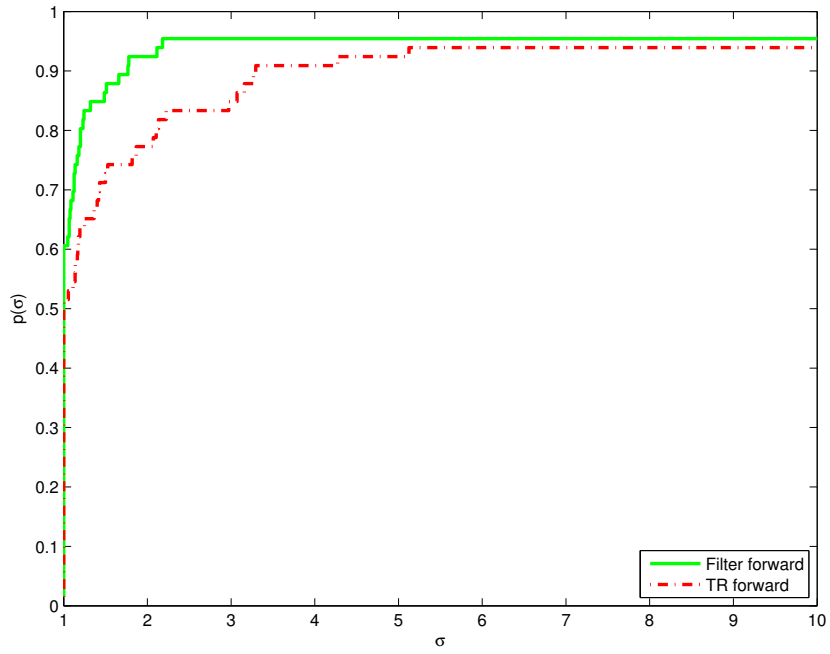


Figure 6.4: CPU time performance profile with approximate second derivatives by forward finite differences when the gradient is available

mula are stated in Table 6.3, while those for a central finite-difference formula are given in Table 6.4.

| | stepsize for $\nabla_x f$ | stepsize for $\nabla^2_{xx} f$ |
|---|---|---|
| stepsize 1 | $\sqrt{\epsilon_M}$ | $\sqrt[4]{\epsilon_M}$ |
| stepsize 2 | $\sqrt{\epsilon_M}$ | $\sqrt[4]{\epsilon_M}\max(\lvert x_j\rvert, 1)$ |
| stepsize 3 | $\sqrt{\epsilon_M}$ | sign $(x_j)\sqrt[4]{\epsilon_M}\max(\lvert x_j\rvert, 1)$ |
| stepsize 4 | $\sqrt{\epsilon_M}$ | $10^{-4}(1 + \lvert x_j\rvert)$ |
| stepsize 5 | sign $(x_j)\sqrt{\epsilon_M}\max(\lvert x_j\rvert, 1)$ | sign $(x_j)\sqrt[4]{\epsilon_M}\max(\lvert x_j\rvert, 1)$ |

Table 6.3: The different stepsizes for a *forward finite-difference* approximation to the gradient and a finite-difference approximation to the Hessian using only function values

The stepsizes that give the best performances on our test problem set are the third one when the first derivatives are approximated by forward finite differences and again the third one for

| | stepsize for $\nabla_x f$ | stepsize for $\nabla^2_{xx} f$ |
|---|---|---|
| stepsize 1 | $\sqrt[3]{\epsilon_M}$ | $\sqrt[4]{\epsilon_M}$ |
| stepsize 2 | $\sqrt[3]{\epsilon_M}$ | $\sqrt[4]{\epsilon_M}\max(\|x_j\|, 1)$ |
| stepsize 3 | $\sqrt[3]{\epsilon_M}$ | sign $(x_j)\sqrt[4]{\epsilon_M}\max(\|x_j\|, 1)$ |
| stepsize 4 | sign $(x_j)\sqrt[3]{\epsilon_M}\max(\|x_j\|, 1)$ | sign $(x_j)\sqrt[4]{\epsilon_M}\max(\|x_j\|, 1)$ |
| stepsize 5 | $10^{-4}(1 + \|x_j\|)$ | $10^{-4}(1 + \|x_j\|)$ |

Table 6.4: The different stepsizes for a *central finite-difference* approximation to the gradient and a finite-difference approximation to the Hessian using only function values

central finite differences. Figures 6.5 and 6.6 give the iteration and CPU time performance plots for the filter and the pure trust-region variants with exact derivatives, and first and second derivatives approximated by finite differences.

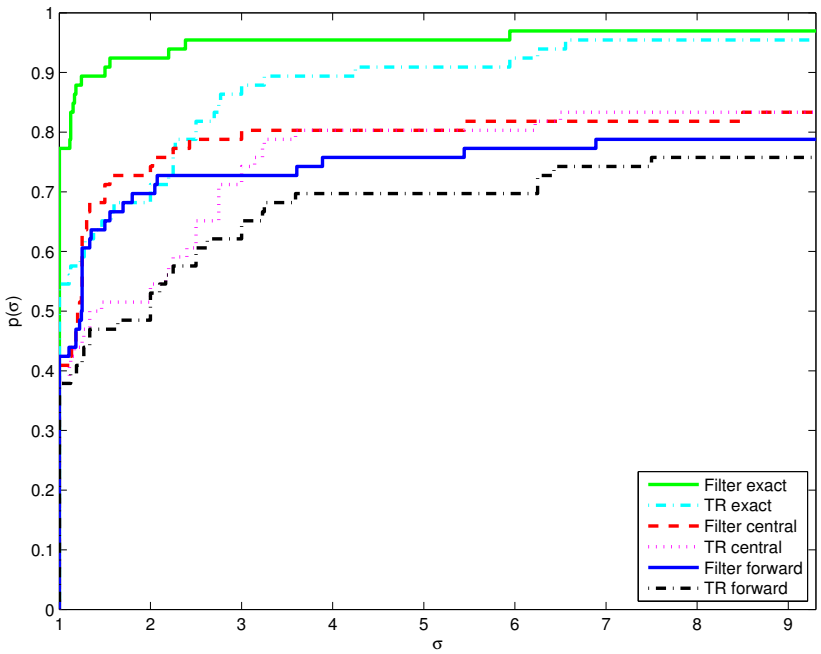


Figure 6.5: Iteration performance profile with exact derivatives and approximate first and second derivatives by finite differences

As expected in Section 1.3.4, the variants with both derivatives approximated by finite differences using only function values pay a heavy price in computing time and are less efficient and robust than those where the gradient is available. This CPU time penalty is, for a part, due to the fact that more iterations are now needed but also to the fact that lots of function evalua-

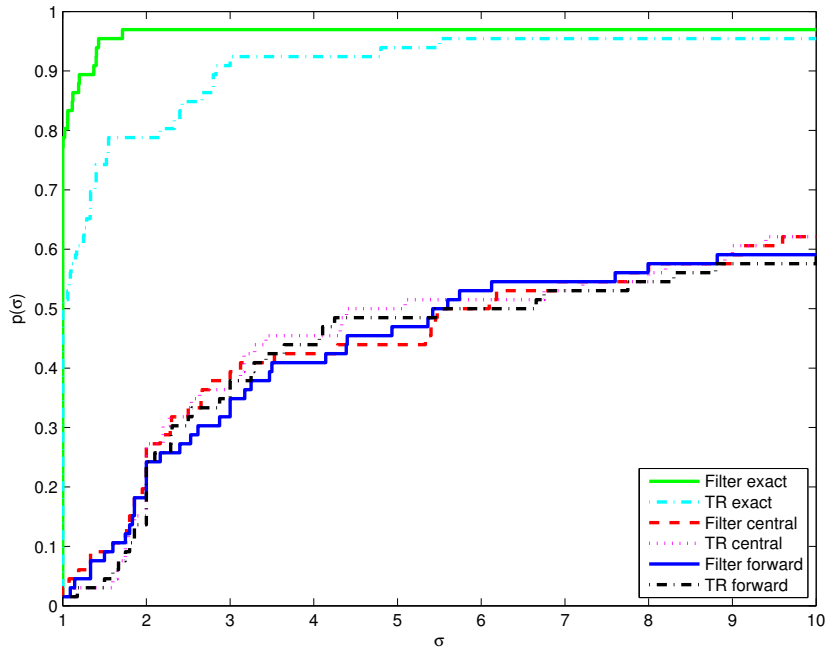


Figure 6.6: CPU time performance profile with exact derivatives and approximate first and second derivatives by finite differences

tions are required to approximate the gradient and the Hessian.
Since the first derivative is exactly the criticality measure we are trying to drive to zero, these results prove its importance in our algorithm as in other algorithms for unconstrained minimization. However, we can point out the fact that, even with these approximations to both first and second derivatives, each filter variant is significantly better than its corresponding pure trust-region variant as well as regards number of iterations as for CPU time. This can be distinctly observed in Figures 6.7 and 6.8.

Finally, we would like to remind our readers that, when both derivatives are not available or are time-consuming, it is generally preferable to use DFO techniques. Derivative-free optimization is concerned with the study of mathematical programs for which it is assumed that the first and (a fortiori) the second derivatives of the functions involved in the problem are not available. DFO methods generally approximate the objective function without approximating its derivatives. For complete surveys on this topic, we refer the reader to the papers by Conn, Scheinberg and Toint [35, 36], Lewis, Torczon and Trosset [106] and Wright [128]. In his PhD thesis [26], Colson has proposed a derivative-free algorithm for unconstrained optimization and a filter-SQP algorithm for the solution of derivative-free constrained problems. Audet and Dennis [3] also use the filter in the framework of derivative-free optimization.

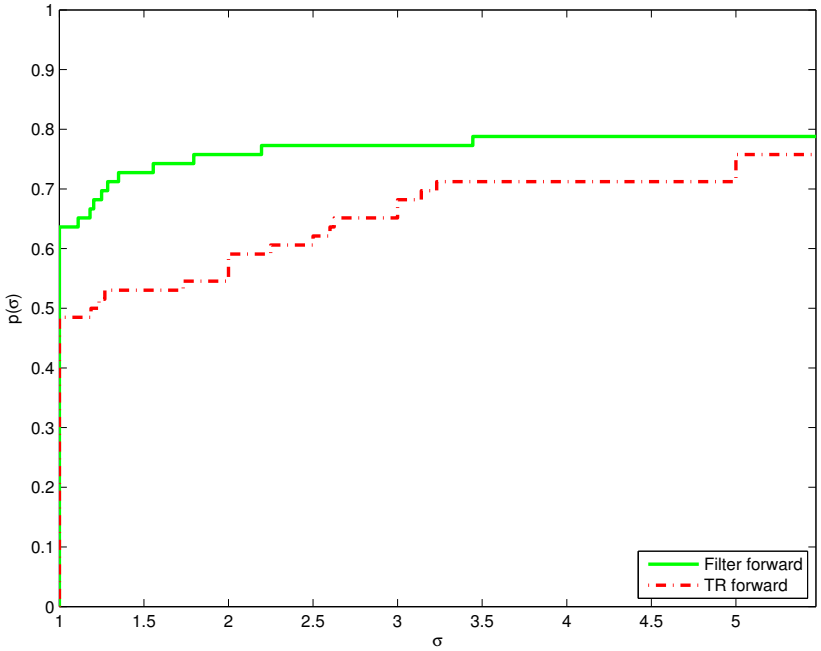


Figure 6.7: Iteration performance profile with approximate first and second derivatives by finite differences

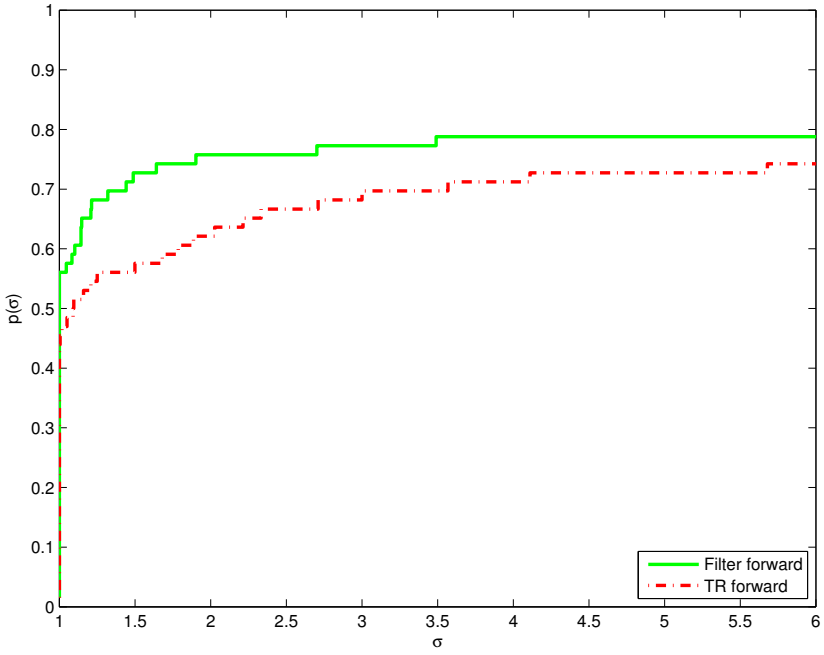


Figure 6.8: CPU time performance profile with approximate first and second derivatives by finite differences

### 6.2.2 Secant updates

We now present numerical experiments with secant updates to Hessian matrices : these techniques are advantageous due to the absence of second-order derivatives computation and additional gradient evaluations. In this section we consider the BFGS and the SR1 secant updating formulae which are recalled in (2.14) and (2.15) (see Dennis and Schnabel [41, Chapter 9] for further details). We have already discussed in Section 2.3.1 different choices for the initial matrix $B_0$ and indicated that there is no choice that can be made to work well in all cases.

#### BFGS updating formula

For the BFGS update, it is not convenient to set $B_0$ to a finite-difference approximation to $\nabla^2_{xx} f(x_0)$. The reason is that secant methods based on this updating formula perform on beginning with and keeping a positive definite Hessian approximation. But with an initial approximation matrix computed by finite differences, there is no guarantee that $B_0$ will be positive definite. So, if one wants to use this initial approximation, the approximate matrix must be perturbed into a positive definite one by applying, for instance, the techniques described in Section 5.5 of Dennis and Schnabel [41]. However, it is interesting to note that, in practice, our filter-trust-region algorithm using BFGS updates behaves very well with an initial finite-difference approximation without correction. But applying BFGS techniques with an indefinite starting matrix is not standard in the literature, although the use of a trust region allow for this. On the other hand, a version with a correction of the initial matrix to be positive definite does not yield better results than simply setting the initial matrix to the identity and implies a higher cost in function and/or gradient evaluations. Therefore the results presented below are obtained by setting $B_0 = I$. The plots in Figures 6.9 and 6.10 show the performances in terms of number of iterations and CPU time for this kind of Hessian approximations compared to the exact variants. We have excluded problems where variants do not report the same final objective function value[(2)].

Both BFGS versions are obviously less efficient and robust than the exact ones. It can be seen that the BFGS filter variant and the BFGS pure trust-region one are essentially comparable as well as in terms of iteration count as in CPU time, indicating that using a BFGS approximation of the Hessian matrix in our filter-trust-region algorithm implies a larger increase in the number of iterations (and thus in the computation time) than using the same approximation within a classical trust-region scheme. However, it can be remarked that the BFGS variants are more competitive in terms of CPU time efficiency with the exact ones than the finite-difference

[(2)]Problems `BIGGS6`, `GROWTHLS`, `GULF` and `JENSMP` have been excluded.

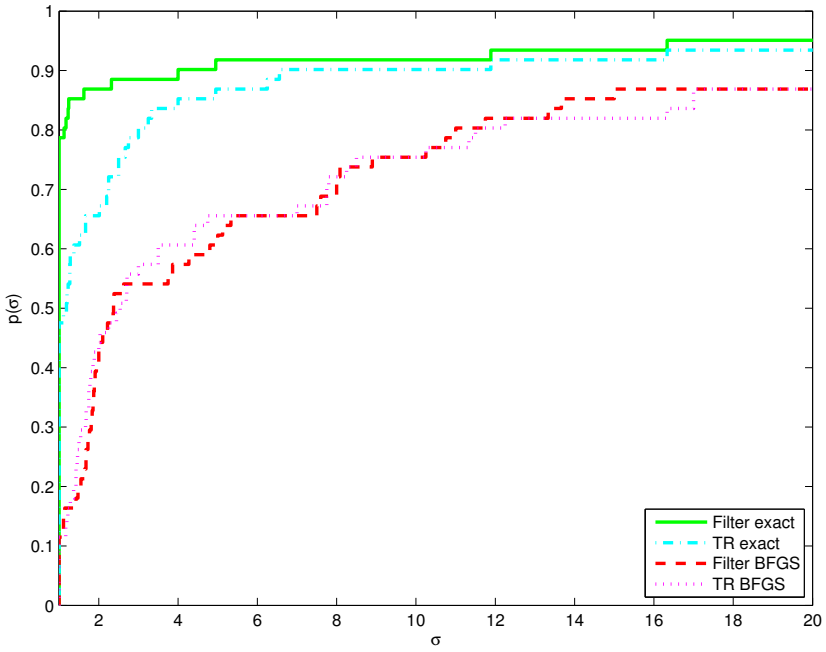


Figure 6.9: Iteration performance profile with exact derivatives and BFGS updates

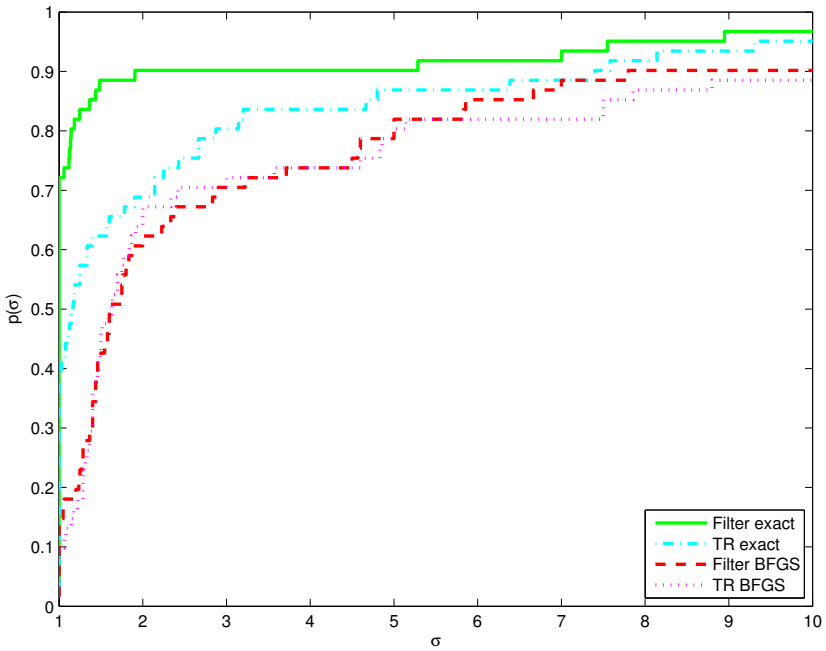


Figure 6.10: CPU time performance profile with exact derivatives and BFGS updates

variants were. It is important to mention that, since the BFGS update only produces positive definite matrices, the flag NONCONVEX in Algorithm 4.1 is never set because negative curvature is never detected while solving the trust-region subproblem. This simplifies our test acceptance mechanism in Step 3 of the algorithm, and, furthermore, the filter is never reinitialized to the empty set. We think that, as our algorithm directly relies on detecting negative curvature, it is more appropriate to use an updating formula which allows to generate indefinite Hessian approximations, like the symmetric-rank-one update. This should indicate that the ability of using nonconvex models for the computation of the trial step might be important in our filter-trust-region algorithm.
Finally, we want to mention that the BFGS update is not very often skipped. A closer look at the results shows that, among successfully solved problems, the BFGS update is never skipped except for problems listed in Table 6.5.

| problem | # of skips |
|---|---|
| AIRCRFTB | 1 |
| BIGGS6 | 1 |
| BOX2 | 8 |
| BROWNBS | 1 |
| DENSCHNC | 61 |
| DENSCHND | 1 |
| DENSCHNE | 2 |
| DJTL | 35 |
| GROWTHLS | 1 |
| HAIRY | 14 |
| HEART8LS | 1 |
| HIELOW | 1 |
| HUMPS | 42 |
| JENSMP | 1 |

Table 6.5: The number of times the BFGS update is skipped

**SR1 updating formula**

We thus consider the SR1 update given in (2.15). As with the BFGS approach, the SR1 update builds an approximate Hessian using gradient information. Nevertheless, unlike the BFGS update, the SR1 Hessian approximation is not restricted to be positive definite. Therefore we can now estimate the initial approximation $B_0$ by a finite-difference technique as this matrix

does not need to be positive definite. We guess that, for our algorithm, the SR1 approximation may be a better strategy, compared to the BFGS one, if there is many negative curvature in the problem since it may be able to maintain a better approximation to the true Hessian in this case then reflecting nonconvexity if it is present. The best results are obtained with a finite-difference estimation of the Hessian matrix at $x_0$ and these results are displayed in Figures 6.11 and 6.12[(3)].

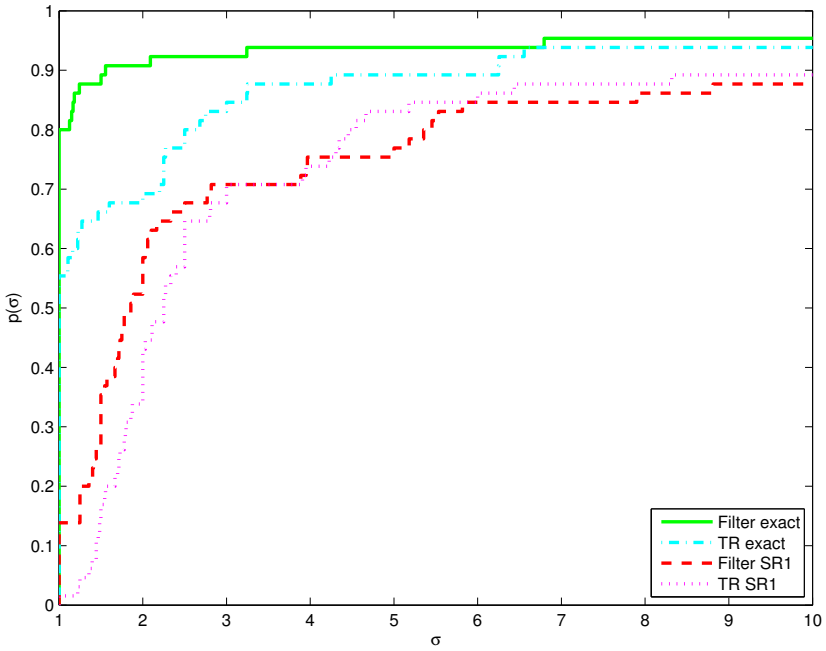


Figure 6.11: Iteration performance profile with exact derivatives and SR1 updates

It can be seen in Figures 6.13 and 6.14, which illustrate the iteration and CPU time performances for both variants with SR1 updates, that the filter variant with SR1 updating formula is a little more efficient both in terms of iterations and CPU time than the corresponding pure trust-region variant, but this latter is a little more reliable. We can deduce from Figure 6.13 that the filter variant with SR1 update is the best solver, in terms of iteration count, on 74% of the problems while the corresponding pure trust-region one is the best in more or less 58% of the cases. For simplicity of presentation, we have truncated the horizontal axis in Figure 6.11, so the right-hand side does not indicate the overall reliability. In fact, the pure-trust-region variant solved 61 problems out of 65 against 60 for the filter variant.

[(3)]We have excluded problem `VIBRBEAM` because all variants do not report the same final objective function value.

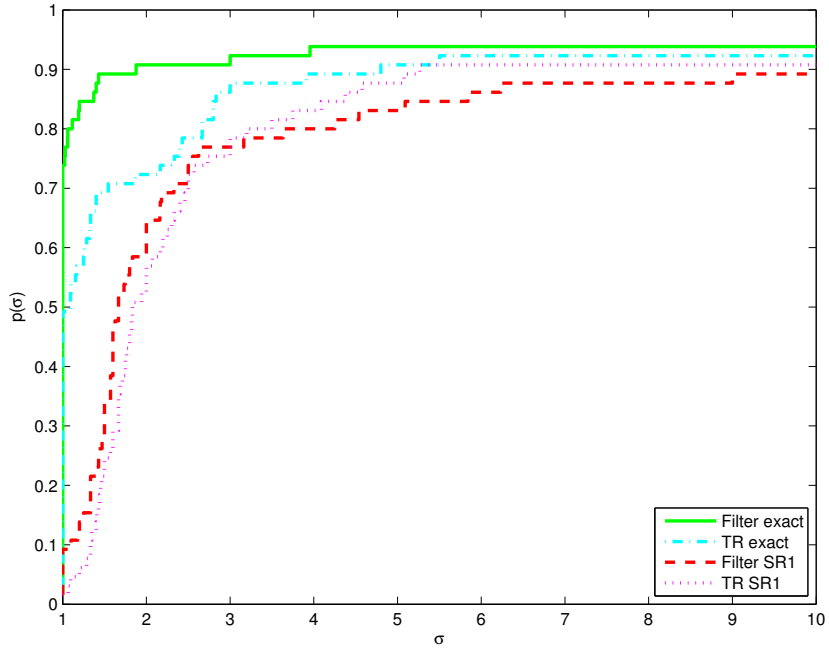


Figure 6.12: CPU time performance profile with exact derivatives and SR1 updates

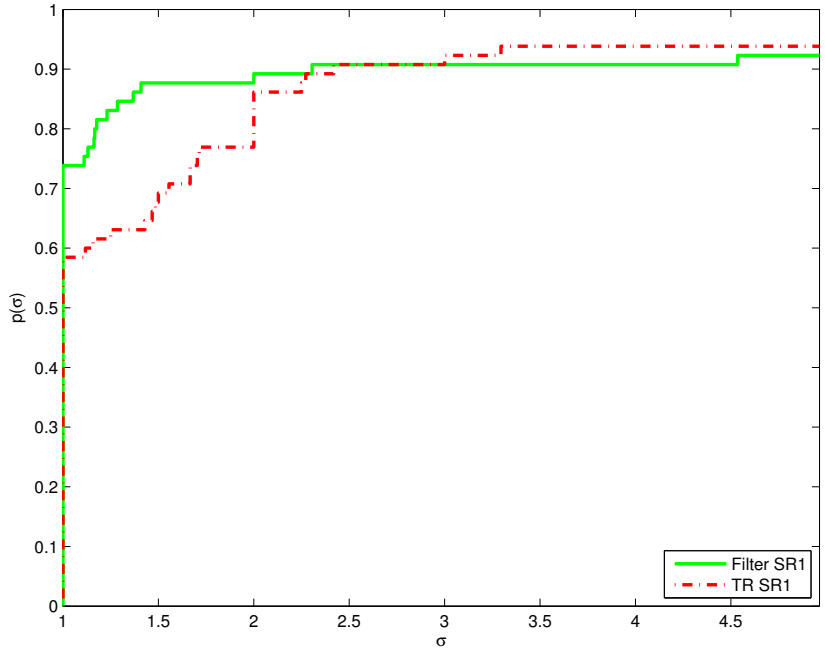


Figure 6.13: Iteration performance profile with the second derivatives approximated by SR1 updating formulae

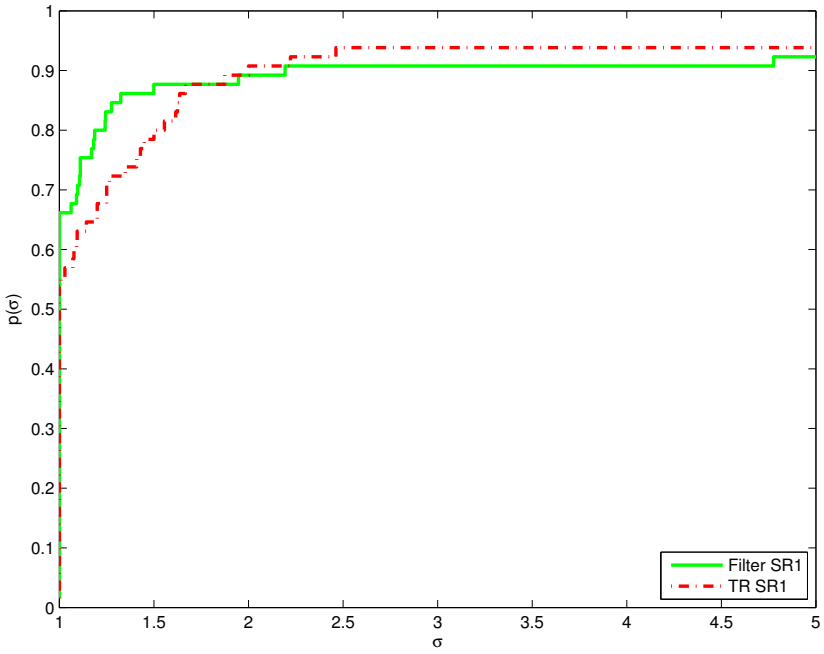


Figure 6.14: CPU time performance profile with the second derivatives approximated by SR1 updating formulae

A closer look at the results shows that, with the SR1 update, the filter is more often reset to the empty set than with exact derivatives[(4)] (while it obviously never happens with BFGS techniques). Table 6.6 gives, for some problems, the iteration count (# iter) and the number of times (# nfilt) the filter is thrown away[(5)] for variants with exact derivatives and with Hessians computed by SR1 updating formulae. It suggests that the SR1 filter variant frequently falls within a region where the updated model is detected to be nonconvex while this is not necessarily the case of the true function. It can be observed that problems for which the filter is very often emptied with SR1 updates also require a larger number of iterations than when they are solved using exact derivatives. Then the ability of SR1 updating techniques to produce nonconvex models may, for some cases, slow down the convergence of the filter-based method.

[(4)]Recall that the filter is reinitialized after each iteration giving sufficient decrease in the objective function $f$ at which the model $m_k$ was detected to be nonconvex.

[(5)]In fact, we count the number of times a negative curvature is encountered on successful iterations, it may happen that an already empty filter has to be reinitialized.

| | exact | | SR1 | |
|---|---|---|---|---|
| Problem | # iter | # nfilt | # iter | # nfilt |
| AIRCRFTB | 21 | 5 | 39 | 11 |
| CUBE | 31 | 0 | 166 | 18 |
| DJTL | 106 | 40 | 215 | 86 |
| ENGVAL2 | 13 | 1 | 36 | 8 |
| GROWTHLS | 160 | 28 | 196 | 64 |
| GULF | 29 | 9 | 115 | 21 |
| HEART8LS | 98 | 53 | 381 | 98 |
| OSBORNEA | 22 | 2 | 128 | 26 |
| OSBORNEB | 16 | 5 | 141 | 30 |
| ROSENBR | 22 | 0 | 114 | 14 |
| SNAIL | 77 | 5 | 245 | 61 |
| YFIT | 42 | 1 | 635 | 19 |

Table 6.6: Number of iterations and number of times the filter is reinitialized

For the SR1 update, we have also considered to reinitialize the Hessian update at the same time as the filter is reset to the empty set. We can see in Figure 6.15 that the variant with this reinitialization is the best in iteration count on 75% of the problems while the variant with a classical SR1 update is the best on 52% of them, and the pure trust-region one on 35% of the cases. So this modification in the implementation of the SR1 updating technique improves the efficiency of the filter variant.

**BFGS versus SR1**

To conclude this section on secant approximations, we show a comparison between filter variants with BFGS or SR1 updating formulae. Both secant approximations are used with the initial matrix set to the identity. Note, in Figure 6.16, that the variant with SR1 updates is more efficient but the BFGS variant is more reliable. On the whole, the filter variant with SR1 updates is better (in terms of iteration count) than the one with BFGS approximations. But, for some problems, like CUBE, DJTL, HAIRY, HATFLDD, OSBORNEA, OSBORNEB, ROSENBR or SINEVAL, the SR1 filter variant requires lots of iterations and, furthermore, for these problems, the number of times the filter is emptied is large. So the choice between BFGS or SR1 updates is sometimes problem-dependent. For instance, the SR1 filter algorithmic variant is by far the best for all PALMER* problems. Figure 6.17 illustrates the CPU time performance of both variants.

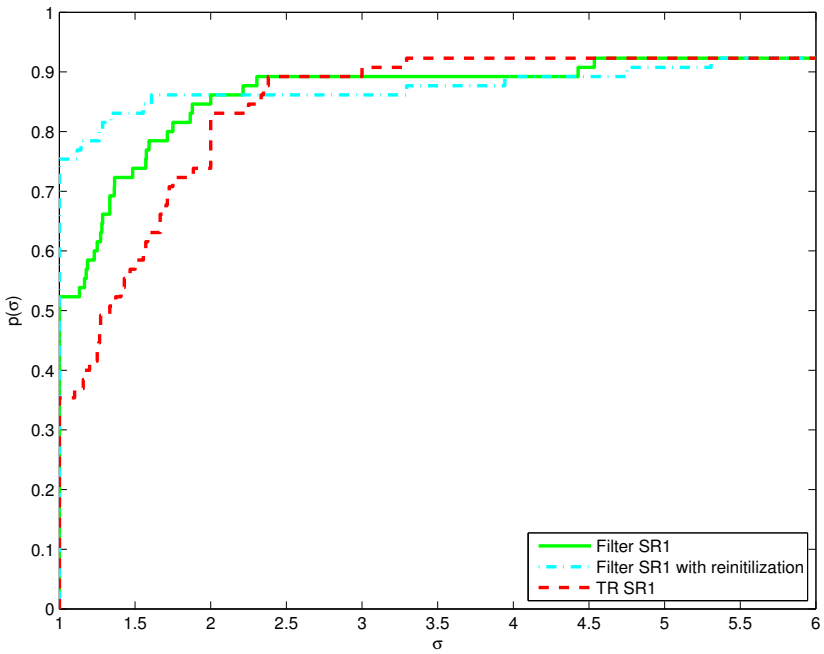


Figure 6.15: Iteration performance profile for filter variants with SR1 updating formulae with reinitialization or not and for the trust-region variant with SR1 updates

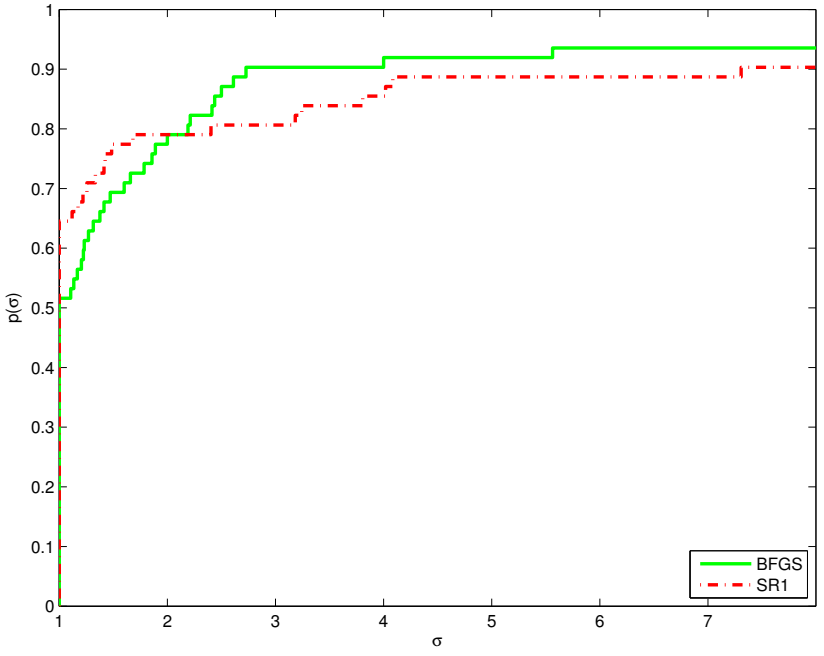


Figure 6.16: Iteration performance profile for the filter variant with BFGS or SR1 updates

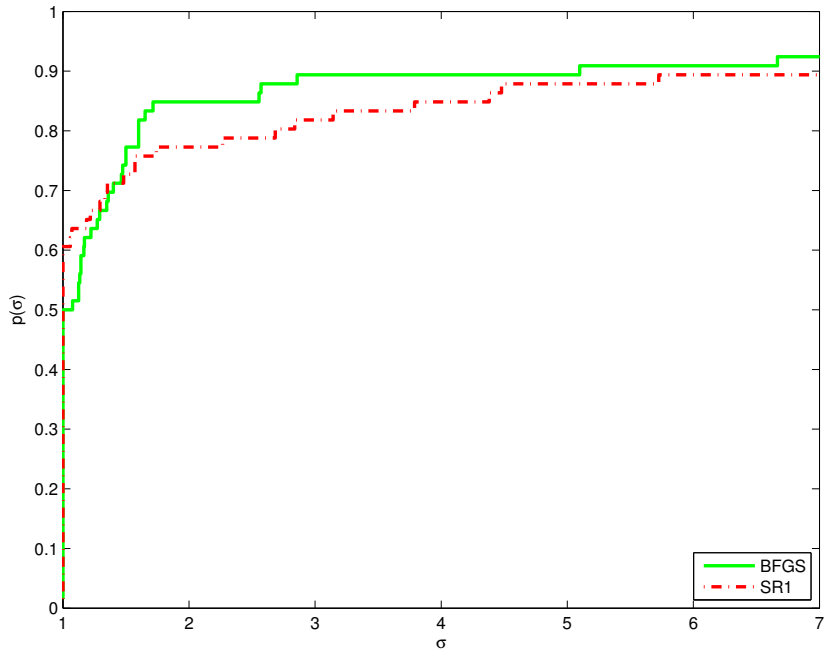


Figure 6.17: CPU time performance profile for the filter variant with BFGS or SR1 updates

### 6.2.3 Perturbation of the Hessian

We have also measured the effect of disturbing the exact Hessian matrices. We will examine two types of perturbation: either we modify all elements of the matrix by a small constant or only the diagonal elements. We have tried $10^{-4}$ and $10^{-6}$ as disturbance values. This sensitivity analysis in terms of number of iterations is shown in Figure 6.18 for $10^{-4}$, where we have only considered the filter variant with either exact or perturbed derivatives.

### 6.2.4 Comparison

We now present a comparison of the different filter variants. As the number of curves would be very high in a performance profile, we prefer to display our comparison by using a combined performance plot (see Gould et al. [67]), where each point consists of the average iteration count and the average CPU time of each tested variant. Note that we only consider problems for which all variants were successful. Therefore, Figure 6.19, which represents the combined performance of all filter tested variants, does not give indication about the robustness of the different variants.

Note that problem `STRATEC`, which is a logit model, has been weeded out when plotting Fig-

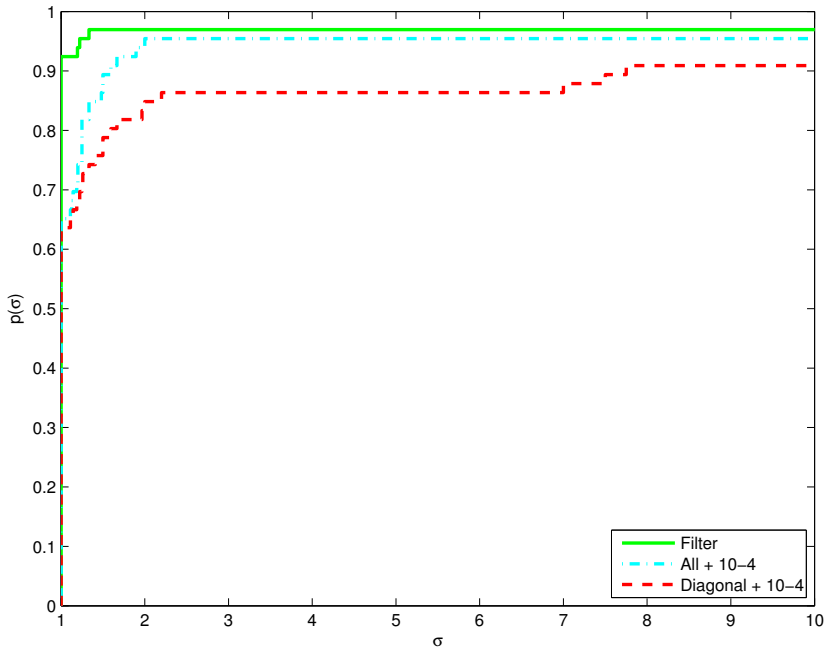


Figure 6.18: Iteration performance profile for the filter variant with exact and perturbed second derivatives

ure 6.19 in order not to mislead the statistics. However, results for this problem will be presented later.

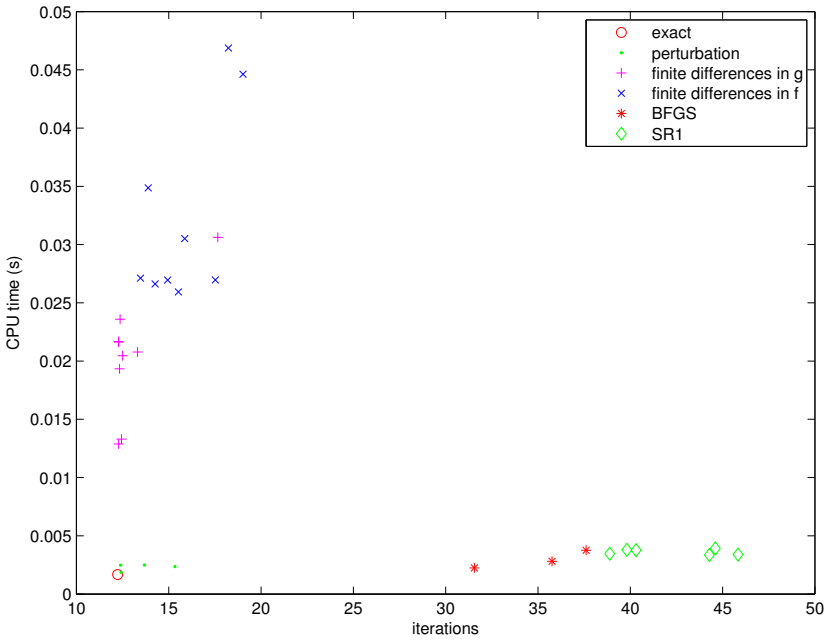


Figure 6.19: Combined performance of all filter variants

It can be clearly seen that the perturbed variants, examined in the previous section, are very close to the variant with exact first and second derivatives. So the disturbance of the Hessian matrix has not a large impact on the performance of the filter-trust-region algorithm.
One can observe that there are two main "clusters". The first one contains the finite-difference variants (with first derivatives available or not, with forward or central formulae, and with different stepsizes); as expected, the variants using these techniques are relatively close to the exact variant in terms of number of iterations but are more computationally expensive. The second cluster includes the secant approximation variants (BFGS and SR1 updates with different initial matrices); these require more iterations but they are very competitive in terms of CPU time with the version using exact derivatives.
As we have excluded problems for which at least one variant has failed, the plot in Figure 6.19 does not give overall information about performance of the variants. For example, BFGS updates are not always better than SR1 ones in terms of iteration count as one may think when observing Figure 6.19.

We now present more accurate results for problem `STRATEC`, which has 10 variables. Again, we only consider the filter variant. As expected by the literature (see, for instance, Bierlaire [9]), Quasi-Newton techniques work well for this kind of problems. It can be noticed in Figure 6.20 that using a secant approximation to the second derivatives implies an important increase of the number of iterations. However, computational time per iteration is drastically reduced, while this measure of time is much higher for the finite-difference versions using gradient values. This is due to the fact that the evaluation of first derivatives is very expensive for this problem. As implemented in the CUTEr distribution, the computation of the gradient is approximately twenty times more expensive than the evaluation of the objective function. Globally, we obtain a significant gain by using secant updates when the evaluations of the derivatives are too expensive. Despite requiring more than 20 additional iterations and more than 80 respectively, the SR1 and BFGS variants remain very competitive from a computational time point of view[(6)].

To conclude this practical study, we reproduce a last plot of some results obtained for problem `HIELOW` which is also a logit model like `STRATEC`. In Figure 6.21 we show the computational time per iteration for the variant with exact derivatives and for two variants with an SR1 update starting with a forward finite-difference approximation to the Hessian but one of them considering the suggestion made in Section 6.2.2, that is the reinitialization of the Hessian each time the filter is reset to the empty set.

[(6)]The SR1 update is performed with the initial matrix estimated by finite differences while the BFGS updating formula is started with the identity.

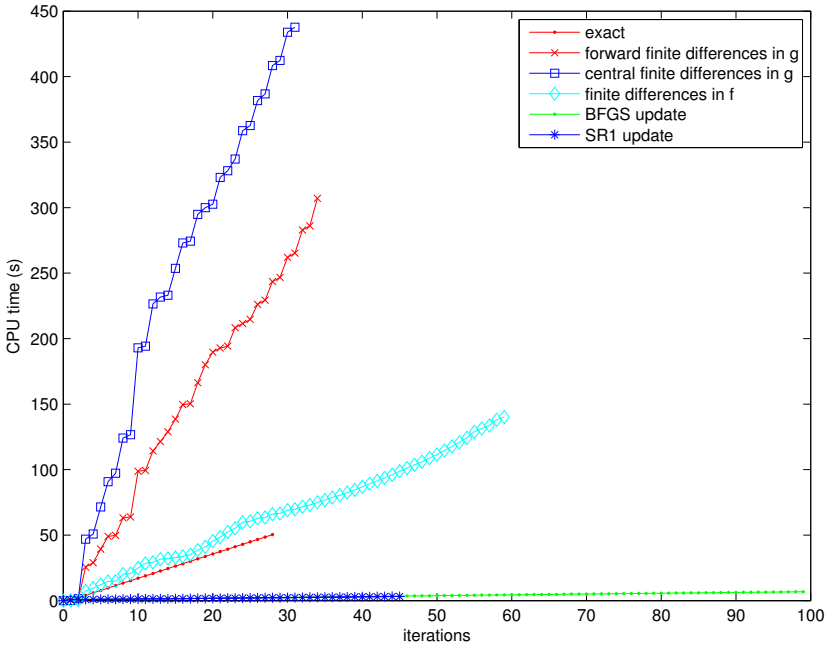


Figure 6.20: Comparison of computational time per iteration between exact and approximate filter variants for problem STRATEC

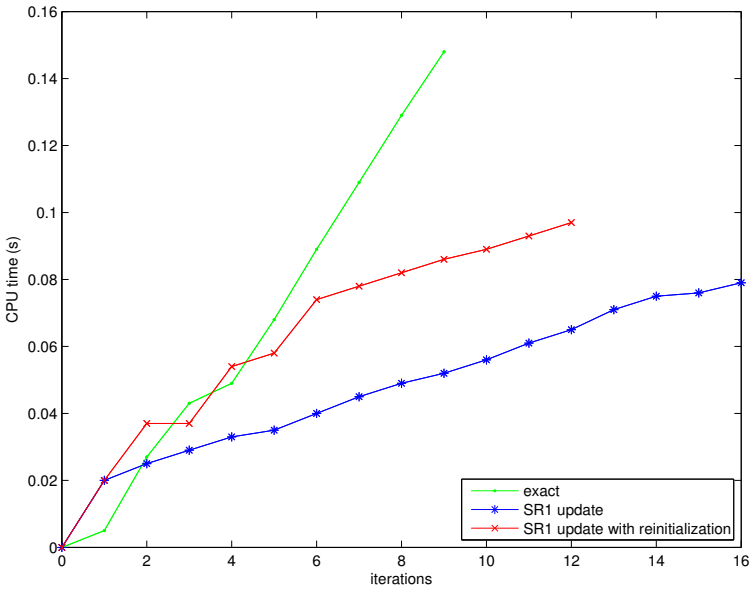


Figure 6.21: Comparison of computational time per iteration between exact and approximate Hessian by SR1 update with reinitialization or not for problem HIELOW

For this problem, the use of a classical SR1 update allows to reduce the computational time per iteration, except for the first iteration where we have to estimate the Hessian by finite differences. While the SR1 update with the reinitialization requires less iterations than the variant without this heuristic, it is more time-consuming because it requires some possible finite-difference computations to reinitialize the Hessian. However, this variant remains less expensive than that using exact derivatives. So, to avoid spending too much time in the computation of the second-order derivatives of such problems, it is generally cheaper to approximate the Hessian using only first-order information (like with the SR1 updates).

## 6.3 Conclusion

We have presented in this chapter a practical study of the influence of approximate derivatives on the filter-trust-region algorithm designed for unconstrained optimization. In view of the numerical experiments carried out on small-scale problems, we can say that the filter-trust-region algorithm does not suffer more than a classical trust-region method from the use of finite-difference approximations to derivatives. But using finite differences to approximate derivatives appears to be justifiable only if the price to perform differencing is slight, and this happens when the cost of evaluating the objective function and its first-order derivatives is also small. As one might expect, the use of exact derivatives in our filter-trust-region algorithm gives in general significantly better results than approximations, both in terms of number of iterations and in terms of total CPU time. So we would recommend our algorithm with exact derivatives when these latter are available. However, for problems where the evaluation of derivatives is costly, like, for instance, complex problems with large data sets, we would recommend a secant update technique. However, in our test set, most problems do not have particularly complicated derivatives, and the computation of the exact Hessian matrix is thus less expensive than performing a secant update.

# Part II

# Bound-Constrained Optimization

# Chapter 7

# A filter-trust-region method for bound-constrained optimization

We now turn to the second particular type of nonlinear optimization problems investigated in this dissertation, namely bound-constrained optimization problems. We propose a theoretical and practical extension of the filter-trust-region algorithm described in Chapter 4 for solving nonlinear and possibly nonconvex optimization problems with simple bounds. The two main ingredients of the method are a filter-trust-region algorithm and a gradient-projection method. Note that the choice of a projection-type method is one of the possible ways to adapt this technique to bound-constrained optimization, but not the only one. We could, for example, use an interior-point method to deal with bounds.

We will start this chapter with a section whose aim is to provide an introduction to simple-bound-constrained optimization and to introduce some tools related to this particular class of mathematical programs. We will describe our method in Section 7.2 while the global convergence to first-order critical points is considered in Section 7.3. The numerical assessment of the proposed method will be reported in Chapter 8. The algorithm described in this chapter as well as its numerical performance analysis were first introduced in the paper of Sainvitu and Toint [111].

## 7.1 Introduction to bound-constrained optimization

Having studied an algorithm for unconstrained optimization problems in the first part of this work, we now consider the special case where the only constraints are restrictions on the

variables. The mathematical program is the following

$$\begin{array}{ll} \min & f(x) \\ \text{s.t.} & l \leq x \leq u, \end{array} \tag{7.1}$$

where $f$ is a twice continuously differentiable function of the variables $x \in \mathbb{R}^n$ and $l$ and $u$ represent lower and upper bounds on the variables. Note that any of these bounds may be infinite. Without loss of generality, we assume that $l_i < u_i$ for all $i = 1, \ldots, n$.

Bound-constrained optimization problems of the form (7.1) have an important role in the design of computer codes for general constrained optimization problems. Indeed, many softwares solve a general constrained mathematical program by finding solutions of a sequence of bound-constrained problems (as seen in Section 2.4). Furthermore, unconstrained problems should involve bounds to avoid bad effects due to computer arithmetic and many real-life problems include simple bounds. For example, it is common, in applications, that parameters which describe physical quantities are constrained to lie in a given range.
Note that we have to make a distinction between *soft* bounds and *hard* bounds. These names illustrate that hard bounds are to be necessarily satisfied, because, for instance, the objective function can not be evaluated outside the feasible set, while soft bounds only express some restrictions we want to impose on the values taken by the variables, but, in these cases, the function is defined outside the feasible region determined by the soft bounds. When constraints involving in the problem are hard ones, feasible-type approaches are to be used, like interior-point methods, because the iterates remain in the interior of the feasible set. On the other hand, treating problems subject to soft constraints is possible, for instance, by using penalty methods. In this dissertation, we consider the bounds to be hard.

### 7.1.1 Optimality conditions

The set of points which satisfy the constraints in problem (7.1) is the *feasible box* and is denoted by

$$\Omega = \{x \in \mathbb{R}^n \mid l \leq x \leq u\}. \tag{7.2}$$

Any point belonging to this box is said to be *feasible*. An *active constraint* indicates here that a variable lies on one of its bounds. The *active set* is then defined as the following set

$$\mathcal{A}(x) \stackrel{\text{def}}{=} \{i \mid x_i = l_i \text{ or } x_i = u_i\}. \tag{7.3}$$

The optimality conditions presented in Section 2.2 are simplified in bound-constrained optimization. The first-order necessary condition can be expressed as the following theorem.

**Theorem 7.1** (First-order necessary condition for a bound-constrained problem)
Suppose that $x^*$ is a local minimizer of problem (7.1) and $f$ is continuously differentiable in an open neighbourhood of $x^*$. Then, defining the *binding set* as

$$\mathcal{B}(x^*) \stackrel{\text{def}}{=} \left\{ i \mid x_i^* = l_i \text{ and } \frac{\partial f(x^*)}{\partial x_i} \geq 0 \right\} \cup \left\{ i \mid x_i^* = u_i \text{ and } \frac{\partial f(x^*)}{\partial x_i} \leq 0 \right\},$$

we have

$$\frac{\partial f(x^*)}{\partial x_i} = 0, \quad i \notin \mathcal{B}(x^*).$$

The latter theorem essentially requires that all partial derivatives of $f$ with respect to $x_i$ which are not at their upper or lower bounds be zero, and those partial derivatives with respect to $x_i$ which are at a bound must be larger than zero at the lower bound and less than zero at the upper one.
Second-order sufficient conditions for $x^*$ to be a local minimizer of problem (7.1) are stated as follows.

**Theorem 7.2** (Second-order sufficient conditions for a bound-constrained problem)
Suppose that the first-order condition (7.1) holds and that, furthermore,

$$s^T \nabla^2_{xx} f(x^*) s > 0 \text{ for all vectors } s,\ s \neq 0,\ s_i = 0 \text{ for } i \in \mathcal{B}_s(x^*),$$

where $\mathcal{B}_s(x^*)$ is known as the *strictly binding set* at $x^*$ and is defined as

$$\mathcal{B}_s(x^*) \stackrel{\text{def}}{=} \mathcal{B}(x^*) \cap \left\{ i \;\middle|\; \frac{\partial f(x^*)}{\partial x_i} \neq 0 \right\}.$$

Then $x^*$ is a local minimizer of problem (7.1).

Considering the set of indices of *free variables* (variables which are not at one of their bounds), which is of the form

$$\mathcal{F}(x) \stackrel{\text{def}}{=} \{ i \mid l_i < x_i < u_i \},$$

the *restricted gradient* and the *restricted Hessian* are, respectively, the gradient and the Hessian of the objective function with respect to free variables, *i.e.* to $x_i$ $(i \in \mathcal{F}(x))$. So algorithms designed for the solution of box-constrained optimization problems typically perform by iden-

tifying these free variables and then using unconstrained minimization methods described in Section 2.3 to explore the corresponding "restricted" problem in order to drive the restricted gradient to zero.

### 7.1.2 Gradient-projection method

Many algorithms developed for bound-constrained problems belong to the class of active-set methods[(1)](see Gill, Murray and Wright [57] or Fletcher [48] for more details). However, one major criticism of classical active-set methods is that the working set changes very slowly at each iteration. Such methods may thus require many iterations to converge on large-scale problems. The *gradient-projection method* (see *e.g.* Conn et al. [28, 29], Lin and Moré [89], Moré [94] and Moré and Toraldo [96]) is designed to allow rapid changes in the active set at each iteration.

Given an iterate $x_k$, naive gradient-projection methods (see, for example, Levitin and Polyak [88], Conn et al. [34] or Nocedal and Wright [101]) search along the *projected-gradient path* or the *piecewise linear path* defined as

$$x_k(t) \stackrel{\text{def}}{=} P[x_k - t\nabla_x f(x_k)],$$

where $t \geq 0$ and $P[\cdot]$ is the projection onto the feasible region. This path is the projection of the steepest-descent direction onto the feasible region. Note that the projection $P$ of any vector $x$ onto the feasible region is extremely easy to compute when the region is a box. So, in presence of simple-bound constraints, the projection operator, denoted by $P[\cdot, l, u]$, can be defined componentwise by

$$P[x, l, u]_i \stackrel{\text{def}}{=} \begin{cases} l_i & \text{if } \; x_i \leq l_i, \\ x_i & \text{if } \; l_i < x_i < u_i, \\ u_i & \text{if } \; u_i \leq x_i. \end{cases} \tag{7.4}$$

A projected-gradient path is illustrated in Figure 7.1 by the bold line.

A new point

$$x_{k+1} = P[x_k - t_k \nabla_x f(x_k), l, u]$$

is obtained when an appropriate $t_k > 0$ is found. The gradient-projection algorithm ensures that the active set at a solution is determined in a finite number of iterations (see Bertsekas [7]). Once the active set is identified, this method reduces to the simple steepest-descent method on the

[(1)]Basic ideas of active-set methods have been described in Section 2.4.3 for sequential quadratic programming.

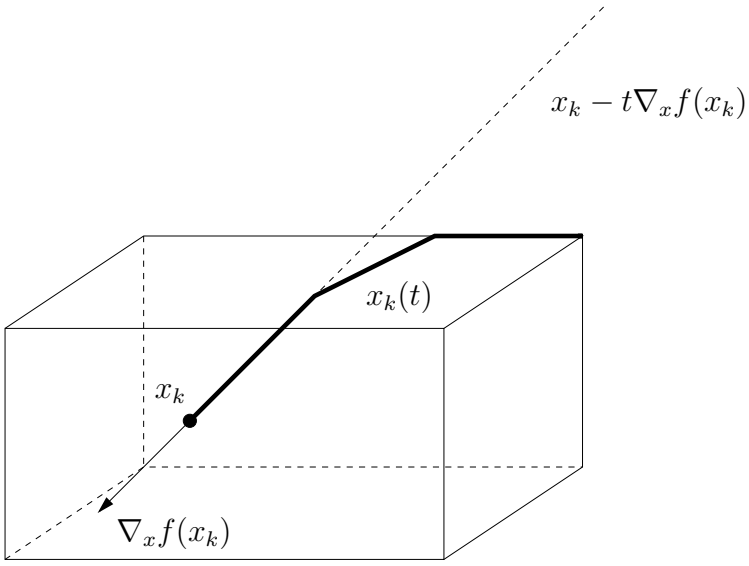


Figure 7.1: A projected-gradient path $x_k(t)$ in $\mathbb{R}^3$

subspace of free variables. Because of the poor rate of convergence of this latter technique, the gradient-projection method described above is usually combined with other algorithms having faster rate of convergence (see Section 7.2.1).

The globalization technique presented in Section 2.3.3, namely the trust region, can also be extended to bound-constrained optimization problems. Within a trust-region scheme, the trial step $s_k$ is now computed by (possibly only approximately) solving the following trust-region subproblem

$$\begin{aligned} \min \quad & m_k(x_k + s) \\ \text{s.t.} \quad & \|s\| \leq \Delta_k \\ & l \leq x_k + s \leq u, \end{aligned}$$

where $\|\cdot\|$ is a suitably chosen norm and $m_k(x_k + s)$ is defined in (2.17). We will describe this approach in detail in the following section.

Note that it is possible to use interior-point methods as an alternative to gradient-projection methods for solving bound-constrained optimization problems.

There are several robust and efficient softwares available for the bound-constrained optimization. LANCELOT (Conn, Gould and Toint [33]) is a gradient-projection method in a trust-region framework and combined with a conjugate-gradient algorithm to accelerate the conver-

gence. TRON developed by Lin and Moré [89] is a trust-region Newton method; it also uses a gradient-projection method combined with a preconditioned conjugate-gradient method. The LBFGS-B software (see Zhu et al. [131]) is a limited-memory Quasi-Newton code for large-scale bound-constrained or unconstrained optimization and uses a line-search technique. Other reliable codes, as IPOPT, KNITRO and LOQO, can also solve bound-constrained problems.

## 7.2 The new algorithm

In this section we present a new filter-trust-region algorithm for the solution of optimization problems subject to simple bounds. To this end, we need to define some concepts.
The *"projected" gradient* of the objective function $f$ onto the feasible box (7.2) is defined by

$$\bar{g}(x) \stackrel{\text{def}}{=} x - P[x - \nabla_x f(x), l, u], \tag{7.5}$$

where the operator $P[\cdot, l, u]$ is defined in (7.4). This projected gradient can be used to characterize first-order critical points; a point $x^* \in \Omega$ is a first-order critical point for problem (7.1) if and only if

$$\bar{g}(x^*) = 0. \tag{7.6}$$

In what follows, we will use the following *first-order criticality measure* of the $k$th iterate

$$\pi(k, x_k) = \pi(x_k) \stackrel{\text{def}}{=} \|x_k - P[x_k - \nabla_x f(x_k), l, u]\|_\infty = \|\bar{g}(x_k)\|_\infty \tag{7.7}$$

(see *e.g.* Conn, Gould and Toint [34, Chapter 8] and [28]). The nonnegative real function defined in (7.7) is a criticality measure because of its continuity (see Section 2.2.1).

In this second part of the dissertation, we propose a extension of the filter-trust-region algorithm presented in Chapter 4 to cover the bound-constrained case. The major ingredients of the proposed algorithm are :

- a *trust-region* framework (see Section 2.3.3);
- a *multidimensional filter* technique (see Section 4.1.2);
- a *gradient-projection* method (see Section 7.1).

In the context of bound-constrained optimization, the optimality condition (7.6) suggests that an iterative method for solving the problem (7.1) must drive the projected gradient $\bar{g}(x_k)$ to

zero for some sequence of feasible $x_k$'s. Therefore, the aim of the filter here is to encourage convergence to first-order critical points by driving every component of the projected gradient

$$\bar{g}(x) = (\bar{g}_1(x), \bar{g}_2(x), \ldots, \bar{g}_n(x))^T$$

to zero. So each entry of the multidimensional filter is a component of $\bar{g}(x)$. A filter is represented in Figure 7.2 for a two-dimensional setup, *i.e.* $x \in \mathbb{R}^2$.

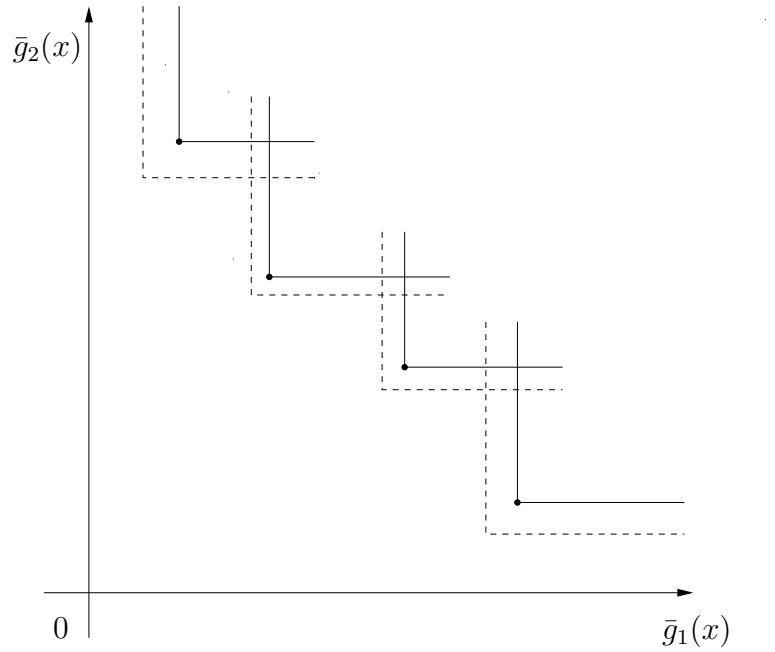


Figure 7.2: A filter for a bound-constrained problem in $\mathbb{R}^2$

### 7.2.1 Computing a trial point

The focus of this section is to describe how to compute the trial point

$$x_k^+ = x_k + s_k$$

from a current feasible iterate $x_k$. We now use the $\ell_\infty$-norm to define the trust region

$$\mathcal{B}_k = \{x_k + s \mid \|s\|_\infty \leq \Delta_k\}.$$

A trial step $s_k$ is then computed by finding an approximation to the solution of the following trust-region subproblem

$$\begin{array}{ll} \text{minimize} & m_k(x_k + s) \\ \text{subject to} & l - x_k \leq s \leq u - x_k \\ & \|s\|_\infty \leq \Delta_k. \end{array} \tag{7.8}$$

As suggested in Section 7.1, solving this subproblem could be achieved by using a gradient-projection method to identify the set of active bounds, $\mathcal{A}(x)$, followed by a minimization of the quadratic model over the subspace of remaining free variables (*i.e.* $x_i$ such that $i \notin \mathcal{A}(x)$). The geometry of the "box" shapes of the $\ell_\infty$-norm and of the simple bounds may be simply exploited. We can rewrite the bounds in (7.8) by the following "box" constraints

$$l_{k,i} \stackrel{\text{def}}{=} \max\left(l_i - x_{k,i}, -\Delta_k\right) \leq s_i \leq \min\left(u_i - x_{k,i}, \Delta_k\right) \stackrel{\text{def}}{=} u_{k,i} \quad \forall\, i = 1, \ldots, n. \tag{7.9}$$

As for our filter-trust-region algorithm for unconstrained optimization, we do not require the step to lie within the trust region at every iteration of our algorithm.

At each iteration, the solution of the trust-region subproblem (7.8) is thus achieved in two stages. At the first one, we use the gradient-projection method to compute the Generalized Cauchy Point (GCP) (or only an approximation to it), denoted by $x_k^C$. As it is common in trust-region methods [34, Chapter 6 and 8], the convergence analysis of Section 7.3 requires that the step provides, at every iteration $k$, a *sufficient decrease on the model* of the objective function. So we need to find a feasible point that gives as much reduction in the model $m_k$ as the generalized Cauchy point, that is

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{\text{mdc}} \pi_k \min\left[\frac{\pi_k}{\beta_k}, \Delta_k\right], \tag{7.10}$$

where $\kappa_{\text{mdc}}$ is a constant in $(0, 1)$, $\pi_k \stackrel{\text{def}}{=} \pi(x_k)$ and

$$\beta_k \stackrel{\text{def}}{=} 1 + \|H_k\|. \tag{7.11}$$

In order to compute the generalized Cauchy point (see Conn, Gould and Toint [29], Moré [94] or Toint [120]), we consider the generalization of the *Cauchy arc* $x_k^C(t)$ (2.21)

$$x_k^C(t) \stackrel{\text{def}}{=} \{x \mid x = P[x_k - t g_k, l_k, u_k],\ t \geq 0\},$$

where $g_k \stackrel{\text{def}}{=} g(x_k) = \nabla_x f(x_k)$. Note that this definition depends on the trust-region boundary because of (7.9). The scalar $t_k$ that determines the GCP $x_k^C$ is chosen so that the Cauchy step

$$s_k^C = x_k^C - x_k$$

produces a sufficient reduction of the objective model. The *generalized Cauchy point* $x_k^C$ is then defined as the first local minimizer of the univariate, piecewise quadratic function

$$m_k(x_k^C(t)), \qquad \text{for } t \geq 0,$$

that is the first local minimizer of the model of the objective function along the Cauchy arc $x_k^C(t)$, which is is continuous and piecewise linear. The generalized Cauchy point is thus computed by investigating the model behaviour between successive pairs of breakpoints, that are points at which a bound is encountered along the Cauchy arc, until the model starts to rise. So further progress along the boundary of the trust region is thus only possible if the model continues to reduce. The variables which lie on one of their bounds at the GCP are fixed thereafter. This process to calculate the GCP allows to add and drop many bounds from the active set defined in (7.3) at each iteration, which can be important for large-scale problems.
There are efficient numerical algorithms for the computation of the generalized Cauchy point which ensure that the model reduction (7.10) is satisfied. We refer the reader to Conn, Gould and Toint [29] and [34, Section 17.3] or Nocedal and Wright [101, Section 16.6] for a detailed description of an algorithm for finding the GCP. Lin and Moré [89] have considered to compute an approximation to the generalized Cauchy point, which is sufficient to ensure convergence.

However, this choice for the step may lead to slow convergence, as the method simply reduces to a version of the steepest-descent method. A further reduction of the quadratic model $m_k$, beyond that guaranteed by the generalized Cauchy point, is often desirable if fast convergence is sought. Therefore, at the second stage of the step computation, attempts are made to further reduce the quadratic model $m_k$ by modifying the values of the remaining free variables, that is, those which are not fixed at one of their bounds at the end of the GCP computation. This could be achieved, for instance, by applying a conjugate-gradient algorithm (see Section 2.3.4), starting from the generalized Cauchy point, to the subproblem (7.8) with the additional restriction that the variables fixed at the GCP remain fixed throughout the process (see Andretta et al. [1], Conn et al. [29, 34], Dostál [44] or Lin and Moré [89]). This conjugate-gradient process is stopped if one of the following occurrences happens :

- the norm of the restricted gradient of the quadratic model with respect to the variables which are not fixed at the generalized Cauchy point, denoted by $(\nabla_x m_k(x_k + s))_{\text{free}}$, falls below some user-defined tolerance, that is

  $$\|(\nabla_x m_k(x_k + s))_{\text{free}}\| \leq \varepsilon_k,$$

  where some value for $\varepsilon_k$ will be given in Section 8.2;

- some fixed maximum number of conjugate-gradient iterations is exceeded;

- at least one of the remaining free variables violates one of the bounds defined in (7.9).

Note that, in the latter case, if the bound encountered is a problem bound and not a trust-region one, we continue the minimization of the model; otherwise, the process is terminated at the point where the boundary is encountered. Our strategy is the following : if a free variable has encountered a true bound of the problem, this offending variable is fixed to the bound and we restart the conjugate-gradient algorithm. We have decided to stop the conjugate-gradient method when a trust-region bound is encountered because we do not want to spend additional time to find a better solution on this trust-region boundary as it is an artificial bound and not a real bound of the problem.

To summarize, each iteration of the above-described technique used to solve the subproblem consists of choosing a face by the gradient-projection method before exploring that face by the conjugate-gradient algorithm. As already said, the advantage of this kind of methods is that the working set is allowed to change rapidly from one iteration to another, which is especially important for large-scale optimization problems.

### 7.2.2 The multidimensional filter

By contrast to traditional trust-region algorithms, we use, as in our algorithm for unconstrained problems, a filter mechanism to assess the suitability of the trial point $x_k^+$. Our strategy is inspired by that of Chapter 4: we decide that a trial point $x_k^+$ is *acceptable for the filter* $\mathcal{F}$ if and only if

$$\forall\ \bar{g}_\ell \in \mathcal{F} \qquad \exists\ j \in \{1,\ldots,n\} \quad \text{such that} \qquad |\bar{g}_j(x_k^+)| < |\bar{g}_{\ell,j}| - \gamma_{\bar{g}}\|\bar{g}_\ell\|, \tag{7.12}$$

where $\gamma_{\bar{g}} \in (0, 1/\sqrt{n})$ is a small positive constant and where $\bar{g}_{\ell,j} \stackrel{\text{def}}{=} \bar{g}_j(x_\ell)$. We then say that $x_k^+$ is not *dominated* by $x_\ell$. If an iterate $x_k$ is acceptable in the sense of (7.12), we may wish to add it to the *multidimensional filter*, which is a list of $n$-tuples of the form $(\bar{g}_{k,1},\ldots,\bar{g}_{k,n})$, such that none of the corresponding iterates is dominated by any other. We also remove from the filter every $\bar{g}_\ell \in \mathcal{F}$ such that $|\bar{g}_{\ell,j}| \geq |\bar{g}_{k,j}|$ for all $j \in \{1,\ldots,n\}$. We refer the reader to Chapter 4 and the mechanisms described therein for further details.

Again the above-described mechanism is adequate for convex problems because the fact that

$$\bar{g}(x^*) = 0 \ \text{ and then } \ \pi(x^*) = 0$$

is both necessary and sufficient for second-order criticality. However, it may be inappropriate for nonconvex problems for which an increase in the projected gradient components is desirable.

Therefore, as in Chapter 4, we modify the filter mechanism to ensure that the filter is reset to the empty set after each iteration giving sufficient descent on the objective function (in the sense of (7.10)) at which the model $m_k$ was discovered to be nonconvex, and we again set an upper bound on the acceptable objective function values in order to guarantee that the obtained decrease is permanent.

### 7.2.3 The filter-trust-region algorithm

We are now in a position to summarize these ideas into a complete algorithm, where the main goal of the filter is to ensure the convergence when convexity is present and where we resort to a classical trust-region algorithm if negative curvature is encountered during the resolution of the subproblem (7.8) or if things do not go well.

A more detailed setup for our algorithm is given hereunder :

**Algorithm 7.1: Filter-Trust-Region Algorithm for bound-constrained optimization**

**Step 0 : Initialization.** Let be given an initial point $x_0 \in \Omega$ and an initial trust-region radius $\Delta_0 > 0$. The constants $\gamma_{\bar{g}} \in (0, 1/\sqrt{n})$, $\eta_1, \eta_2, \gamma_1$, $\gamma_2$ and $\gamma_3$ are also given and satisfy

$$0 < \eta_1 \leq \eta_2 < 1 \quad \text{and} \quad 0 < \gamma_1 \leq \gamma_2 < 1 \leq \gamma_3.$$

Compute $f(x_0)$ and $\bar{g}(x_0)$, set $k = 0$. Initialize the filter $\mathcal{F}$ to the empty set and choose $f_{\sup} \geq f(x_0)$. Define two flags `RESTRICT` and `NONCONVEX`, the former to be unset.

**Step 1: Determine a trial step.** Compute a finite step $s_k$ such that $x_k + s_k \in \Omega$, that "sufficiently reduces" the model $m_k$, *i.e.* that satisfies (7.10), and that also satisfies $\|s_k\|_\infty \leq \Delta_k$ if `RESTRICT` is set or if $m_k$ is nonconvex. In the latter case, set `NONCONVEX`; otherwise unset it. Compute the trial point $x_k^+ = x_k + s_k$.

**Step 2:** Compute $f(x_k^+)$ and define the following ratio

$$\rho_k = \frac{f(x_k) - f(x_k^+)}{m_k(x_k) - m_k(x_k^+)}.$$

If $f(x_k^+) \geq f_{\sup}$, set $x_{k+1} = x_k$, set `RESTRICT` and go to Step 4.

**Step 3: Tests to accept the trial step.**

- Compute $\bar{g}_k^+ = \bar{g}(x_k^+)$.
- If $x_k^+$ is acceptable for the filter $\mathcal{F}$ and `NONCONVEX` is unset:
  Set $x_{k+1} = x_k^+$, unset `RESTRICT` and add $\bar{g}_k^+$ to the filter $\mathcal{F}$ if either $\rho_k < \eta_1$ or $\|s_k\|_\infty > \Delta_k$.
- If $x_k^+$ is not acceptable for the filter $\mathcal{F}$ or `NONCONVEX` is set:
  If $\rho_k \geq \eta_1$ and $\|s_k\|_\infty \leq \Delta_k$, then

  set $x_{k+1} = x_k^+$, unset `RESTRICT` and if `NONCONVEX` is set, set $f_{\text{sup}} = f(x_{k+1})$ and reinitialize the filter $\mathcal{F}$ to the empty set;

  else set $x_{k+1} = x_k$ and set `RESTRICT`.

**Step 4: Update the trust-region radius.** If $\|s_k\|_\infty \leq \Delta_k$, update the trust-region radius by choosing

$$\Delta_{k+1} \in \begin{cases} [\gamma_1 \Delta_k, \gamma_2 \Delta_k] & \text{if} \quad \rho_k < \eta_1, \\ [\gamma_2 \Delta_k, \Delta_k] & \text{if} \quad \rho_k \in [\eta_1, \eta_2), \\ [\Delta_k, \gamma_3 \Delta_k] & \text{if} \quad \rho_k \geq \eta_2; \end{cases} \tag{7.13}$$

otherwise, set $\Delta_{k+1} = \Delta_k$. Increment $k$ by one and go to Step 1.

As it stands, the algorithm lacks formal stopping criteria. In practice, we obviously stop the calculation if the infinity norm of the projected gradient (7.7) falls below some user-defined tolerance and the flag `NONCONVEX` is unset; or if some fixed maximum number of iterations is exceeded. More details about the practical aspects of the implementation of this algorithm can be found in Section 8.2.

## 7.3 Global convergence to first-order critical points

This section is devoted to the convergence analysis of Algorithm 7.1. We will establish that at least one limit point $x^*$ of the sequence $\{x_k\}$ generated by Algorithm 7.1 is a first-order critical point for problem (7.1). The theoretical results from this section are mostly derived from results known for general trust-region methods [34, Chapter 6] and from those of Section 4.2. We will devote this section to a discussion of the modifications of the convergence analysis of Section 4.2 that are required to cover the bound constrained case, using notably the equivalence between the $\ell_2$- and $\ell_\infty$-norms.

Firstly, we need to make the following assumptions, which are very similar to those established for the convergence analysis of basic trust-region methods (see Conn, Gould and Toint [34, Chapter 6]) and to those stated in the unconstrained case (see Section 4.2.2).

**A1** The objective function $f$ is twice continuously differentiable on $\mathbb{R}^n$.

**A2** The iterates $x_k$ remain in a closed, bounded domain of $\mathbb{R}^n$.

**A3** For all $k$, the model $m_k(x)$ is twice differentiable on $\mathbb{R}^n$ and has a uniformly bounded Hessian.

The combination of these assumptions again implies that there exist constants $\kappa_{\text{l}}$, $\kappa_{\text{u}} \geq \kappa_{\text{l}}$, $\kappa_{\text{ufh}} \geq 1$ and $\kappa_{\text{umh}} \geq 1$ such that

$$f(x) \in [\kappa_{\text{l}}, \kappa_{\text{u}}], \quad \|\nabla^2_{xx} f(x)\| \leq \kappa_{\text{ufh}} \quad \text{and} \quad \|\nabla^2_{xx} m_k(x)\| \leq \kappa_{\text{umh}} - 1 \tag{7.14}$$

for all $k$ and all $x$ in the convex hull of $\{x_k\}$. And again we have that

$$\beta_k \leq \kappa_{\text{umh}} \tag{7.15}$$

for all $k$.
We also restate the sufficient decrease on the model $m_k$ given in (7.10) in the following assumption :

**A4** For all $k$,

$$m_k(x_k) - m_k(x_k + s_k) \geq \kappa_{\text{mdc}} \pi_k \min\left[\frac{\pi_k}{\beta_k}, \Delta_k\right],$$

where $\kappa_{\text{mdc}} \in (0,1)$, $\pi_k \stackrel{\text{def}}{=} \pi(x_k)$ and $\beta_k$ is defined in (7.11).

In what follows, we shall use the same notations and definitions for the set $\mathcal{S}, \mathcal{D}$ and $\mathcal{N}$ as those of Section 4.2.1, but now the set of *filter iterations* is given by

$$\mathcal{A} = \{k \mid \bar{g}_k^+ \text{ is added to the filter } \}.$$

Observe that $\mathcal{A} \subseteq \mathcal{S}$, *i.e.* that $\bar{g}_k^+$ is included into the filter only at successful iterations. We also have that the mechanism of our algorithm imposes that

$$\mathcal{S} \cap \mathcal{N} = \mathcal{D} \cap \mathcal{N}. \tag{7.16}$$

With these definitions, we should derive Lemma 4.1 exactly in the same manner.

We start our convergence analysis to first-order critical points by proving that, as long as a first-order critical point is not approached, we do not have infinitely many successful nonconvex iterations in the course of the algorithm, *i.e.* the set $\mathcal{S} \cap \mathcal{N}$ is always finite. Firstly, we recall two results from Conn, Gould and Toint [34] in order to show that the trust-region radius is bounded away from zero.

The following lemma is the same as Lemma 4.2 except that we now consider a trust region defined by the infinity norm.

**Lemma 7.3** Suppose that A1-A3 hold and that $\|s_k\|_\infty \leq \Delta_k$. Then we have that

$$|f(x_k + s_k) - m_k(x_k + s_k)| \leq \kappa_{\text{ubh}} \Delta_k^2, \tag{7.17}$$

where $x_k + s_k \in \mathcal{B}_k$ and

$$\kappa_{\text{ubh}} \stackrel{\text{def}}{=} n \ \max[\ \kappa_{\text{ufh}}, \kappa_{\text{umh}}]. \tag{7.18}$$

**Proof.** The proof is inspired by [34, Theorem 6.4.1] but, in our context, we need to make the additional assumption that $\|s_k\|_\infty \leq \Delta_k$ explicit (instead of being implicit, in this reference, in the definition of a trust-region step) and we have to use the equivalence between the $\ell_2$- and $\ell_\infty$-norms. □

We next show that the trust-region radius must increase if the current iterate is not first-order critical and the trust-region radius is small enough.

**Lemma 7.4** Suppose that A1-A4 hold and that $\|s_k\|_\infty \leq \Delta_k$. Suppose furthermore that $\bar{g}_k \neq 0$ and that

$$\Delta_k \leq \frac{\kappa_{\text{mdc}} \pi_k (1 - \eta_2)}{\kappa_{\text{ubh}}}. \tag{7.19}$$

Then we have that $\rho_k \geq \eta_2$ and

$$\Delta_{k+1} \geq \Delta_k. \tag{7.20}$$

**Proof.** The proof is the same as for Theorem 6.4.2 in [34] when $\|s_k\|_\infty \leq \Delta_k$ except that we now have to replace $\|g_k\|$ by the criticality measure $\pi_k$ and that we use A4 instead of the model decrease defined in [34, Chapter 6]. The idea of the proof is to show that, as long as the current iterate is not a first-order critical point, and that the radius satisfies (7.19), the

iteration must be very successful, and the trust-region radius is enlarged according to (7.13). □

Consequently, we may now obtain that the trust-region radius cannot become arbitrarily small if the iterates stay away from first-order critical points.

**Lemma 7.5** Suppose that A1-A4 hold and that there exists a constant $\kappa_{\text{lbg}} > 0$ such that $\pi_k \geq \kappa_{\text{lbg}}$ for all $k$. Then there is a constant $\kappa_{\text{lbd}} > 0$ such that

$$\Delta_k \geq \kappa_{\text{lbd}} \tag{7.21}$$

for all $k$.

**Proof.** The proof is by contradiction and uses Lemma 7.4. It is identical to that of Lemma 4.4 except that we use the $\ell_\infty$-norm of the step instead of the $\ell_2$-norm and that we now have to replace $\|g_k\|$ by the criticality measure $\pi_k$. □

We now state the essential result that the number of successful nonconvex iterations must be finite unless a first-order critical point is approached.

**Theorem 7.6** Suppose that A1-A4 hold and that there exists a constant $\kappa_{\text{lbg}} > 0$ such that $\pi_k \geq \kappa_{\text{lbg}}$ for all $k$. Then there can only be finitely many successful nonconvex iterations in the course of the algorithm, *i.e.*

$$|\mathcal{S} \cap \mathcal{N}| < +\infty.$$

**Proof.** The proof is inspired by Theorem 4.5 except that $\|g_k\|$ is replaced by $\pi_k$ and we now use condition (7.10) on the model reduction. □

We now establish the criticality of the limit point of the sequence of iterates when there are only finitely many successful iterations.

**Theorem 7.7** Suppose that A1-A4 hold and that there are only finitely many successful iterations, *i.e.* $|\mathcal{S}| < +\infty$. Then $x_k = x^*$ for all sufficiently large $k$, and $x^*$ is first-order critical, *i.e.*

$$\pi(x^*) = 0.$$

**Proof.** The proof is identical to that of Theorem 4.6 except that we have to replace $\|g_k\|$ by the criticality measure $\pi_k$ and that we use the $\ell_\infty$-norm of the step instead of the $\ell_2$-norm. □

We restrict our attention, for the remainder of this section, to the case where there are infinitely many successful iterations, *i.e.* $|\mathcal{S}| = +\infty$. We start by investigating what happens if the filter is updated an infinite number of times in the course of the algorithm, *i.e.* $|\mathcal{A}| = +\infty$.

**Theorem 7.8** Suppose that A1-A4 hold and that $|\mathcal{A}| = |\mathcal{S}| = +\infty$. Then

$$\liminf_{k\to\infty} \pi_k = 0. \tag{7.22}$$

In other words, there exists a limit point $x^*$ of the sequence $\{x_k\}$ generated by Algorithm 7.1 which is a first-order critical point for problem (7.1).

**Proof.** The proof is the same as for Theorem 4.7 except that $\|g_k\|$ is replaced by $\pi_k$ and that we use the new filter acceptance definition (7.12). The proof is by contradiction. We suppose that, for all $k$ large enough, $\pi_k \geq \kappa_{\text{lbg}}$ for some $\kappa_{\text{lbg}} > 0$. Theorem 7.6 implies that the filter is no longer reset to the empty set for $k$ sufficiently large. By using the filter test acceptance mechanism and our initial assumption, we can derive a contradiction exactly as in Theorem 4.7. □

Consider now the case where the number of iterates added to the filter in the course of the algorithm is finite, that is $|\mathcal{A}| < +\infty$.

**Theorem 7.9** Suppose that A1-A4 hold and that $|\mathcal{S}| = +\infty$ but $|\mathcal{A}| < +\infty$. Then (7.22) holds.

**Proof.** Again the proof is identical to that of Theorem 4.8. □

The preceeding two results show that at least one of the limit points of the sequence of iterates generated by Algorithm 7.1 satisfies the first-order necessary condition. However, as in Chapter 4 (cfr counter-example of Section 4.2.3), this result cannot be improved to obtain that all limit points are first-order critical, that is

$$\lim_{k\to\infty} \pi_k = 0,$$

without an important modification of our algorithm (see the discussion at the end of Section 4.2.3).

## 7.4 Conclusion

In this chapter, we have proposed an algorithm for the minimization of simple-bound-constrained optimization problems. The underlying idea of this algorithm is to combine three tools of nonlinear programming, namely trust regions, gradient-projection methods and filter techniques. We have shown that, under standard assumptions, it produces at least a first-order critical point, irrespective of the chosen starting point. For the second-order convergence analysis, difficulties are expected since one knows that possibly only one limit point is first-order critical. We can hope that, under the additional assumption that there is only one limit point, we should derive properties about the optimal active-set identification since this identification is mostly a geometric consequence of convergence and of using the generalized Cauchy point. But, in the current state of the work, we have not been able to produce a theory for the identification of the optimal active set and thus for the second-order convergence. This does not seem straightforward and this work remains to be done. However, it does not seem to affect the numerical results of our algorithm, which will be discussed in the next chapter.

# Chapter 8

# Numerical results

We conclude this part on bound-constrained optimization with the numerical assessment of the filter-trust-region algorithm described in the previous chapter. In order to investigate the behaviour of our algorithm, we have tested our implementation, named FILTBOUND (FILter Trust-region algorithm for BOUND-constrained optimization), on a set of test problems from the CUTEr collection. Some practical aspects of the implementation will be the subject of Section 8.2. Some algorithmic variants will be discussed in Section 8.3 and, as in Chapter 5, our code will be compared with the software LANCELOT-B, the latter having been introduced in Section 5.4.3. We will end the chapter with a short conclusion on the numerical experience.

## 8.1 Testing environment

From a practical point of view, we again use the CUTEr collection [69] to perform our numerical experiments. We have selected from this test set problems sharing the following characteristics (where * means "anything goes") :

| | | |
|---|---|---|
| Objective function type | : | * |
| Constraints type | : | B (bounds only) |
| Regularity | : | R (regularity) |
| Degree of available derivatives | : | 2 (analytical second derivatives) |
| Problem interest | : | * |
| Explicit internal variables | : | * |
| Number of variables | : | * |
| Number of constraints | : | * |

Table 8.1: The problem selection

These requirements lead to a list of 109 simple-bound constrained problems[(1)]. We did not consider problems whose only constraints are fixed variables because these problems can be efficiently solved by FILTRUNC (see Chapter 5). Problem names and dimensions are given in Table 8.3. Since a variable can be fixed (*i.e.* its upper and lower bounds are equal, and therefore the variable is not allowed to change), we consider the dimension of a problem as the number of variables minus the number of fixed ones. The size of the problems lies between 1 and 11130. Table 8.2 gives the dimension distribution for the bound-constrained problems we have selected.

| # of free variables | # of problems from the CUTEr set |
|---|---|
| $1 \leq n \leq 9$ | 54 |
| $10 \leq n \leq 99$ | 5 |
| $100 \leq n \leq 999$ | 5 |
| $1000 \leq n \leq 9999$ | 38 |
| $10000 \leq n \leq 20000$ | 5 |

Table 8.2: The distribution of box-constrained problem dimension in the CUTEr collection

All tests presented in this chapter were performed in double precision on a workstation with a 3.2 GHz Pentium IV biprocessor and 2 Gbytes of memory under Suse Professional 9.0 Linux and the Lahey Fortran compiler (version L6.10a) with default options. We have limited all attempts to solve the test problems to a maximum of 1000 iterations or 1 hour of CPU time. Again the variability of CPU times for small times is taken into account by repeatedly solving the same problem until a threshold of ten seconds is exceeded and then taking the average time per run.

## 8.2 Practical aspects

In this section, we give a brief outline of implementation aspects of our filter-trust-region algorithm for box-constrained problems. We focus on discussing the choices for the different parameters involved in Algorithm 7.1, the computation of the trial step and some other algorithmic considerations. Some comments given in Section 5.3 are always valid in this context and are therefore not repeated here.

[(1)]Two problems, namely `CHARDIS0` and `HARKERP2`, were removed because they could not be run within the memory limits of the testing machine by any of the codes.

| Problem | $n$ | Problem | $n$ | Problem | $n$ |
|---|---|---|---|---|---|
| 3PK | 30 | JNLBRNGB | 9604 | PALMER5B | 9 |
| ALLINIT | 3 | LINVERSE | 1999 | PALMER5D | 8 |
| BDEXP | 5000 | LOGROS | 2 | PALMER5E | 8 |
| BIGGSB1 | 5000 | MAXLIKA | 8 | PALMER6A | 6 |
| BQP1VAR | 1 | MCCORMCK | 5000 | PALMER6E | 8 |
| BQPGABIM | 46 | MDHOLE | 2 | PALMER7A | 6 |
| BQPGASIM | 50 | MINSURFO | 5002 | PALMER7E | 8 |
| BQPGAUSS | 2003 | NCVXBQP1 | 10000 | PALMER8A | 6 |
| CAMEL6 | 2 | NCVXBQP2 | 10000 | PALMER8E | 8 |
| CHEBYQAD | 100 | NCVXBQP3 | 10000 | PENTDI | 5000 |
| CHENHARK | 5000 | NOBNDTOR | 5184 | PROBPENL | 500 |
| CVXBQP1 | 10000 | NONSCOMP | 5000 | PSPDOC | 4 |
| DECONVB | 61 | OBSTCLAE | 9604 | QR3DLS | 610 |
| EG1 | 3 | OBSTCLAL | 9604 | QRTQUAD | 5000 |
| EXPLIN | 1200 | OBSTCLBL | 9604 | QUDLIN | 5000 |
| EXPLIN2 | 1200 | OBSTCLBM | 9604 | S368 | 8 |
| EXPQUAD | 1200 | OBSTCLBU | 9604 | SCOND1LS | 5000 |
| HADAMALS | 380 | OSLBQP | 8 | SIM2BQP | 1 |
| HART6 | 6 | ODNAMUR | 11130 | SIMBQP | 2 |
| HATFLDA | 4 | PALMER1 | 4 | SINEALI | 1000 |
| HATFLDB | 4 | PALMER1A | 6 | SPECAN | 9 |
| HATFLDC | 25 | PALMER1B | 4 | TORSION1 | 5184 |
| HIMMELP1 | 2 | PALMER1E | 8 | TORSION2 | 5184 |
| HS1 | 2 | PALMER2 | 4 | TORSION3 | 5184 |
| HS110 | 200 | PALMER2A | 6 | TORSION4 | 5184 |
| HS2 | 2 | PALMER2B | 4 | TORSION5 | 5184 |
| HS25 | 3 | PALMER2E | 8 | TORSION6 | 5184 |
| HS3 | 2 | PALMER3 | 4 | TORSIONA | 5184 |
| HS38 | 4 | PALMER3A | 6 | TORSIONB | 5184 |
| HS3MOD | 2 | PALMER3B | 4 | TORSIONC | 5184 |
| HS4 | 2 | PALMER3E | 8 | TORSIOND | 5184 |
| HS45 | 5 | PALMER4 | 4 | TORSIONE | 5184 |
| HS5 | 2 | PALMER4A | 6 | TORSIONF | 5184 |
| JNLBRNG1 | 9604 | PALMER4B | 4 | WEEDS | 3 |
| JNLBRNG2 | 9604 | PALMER4E | 8 | YFIT | 3 |
| JNLBRNGA | 9604 | PALMER5A | 8 | | |

Table 8.3: The test problems and their dimension

First of all, note that we always use the starting point supplied with the problem in the CUTEr collection. However, if this initial point is not feasible, we project it onto the feasible box (7.2). For the initial trust-region radius, we have chosen $\Delta_0 = 1$. The values of the parameters involved in Algorithm 7.1 used in our implementation are

$$\gamma_1 = 0.0625,\ \gamma_2 = 0.25,\ \gamma_3 = 2.0,\ \eta_1 = 0.01,\ \eta_2 = 0.9,$$

and

$$\gamma_{\bar{g}} = \min\left[0.001, \frac{1}{2\sqrt{n}}\right].$$

We also choose

$$f_{\text{sup}} = \min(10^6|f(x_0)|, f(x_0) + 1000)$$

at Step 0 of the algorithm.

At each iteration of our code, the trial point is computed by approximately minimizing the subproblem (7.8). As explained in Section 7.2.1, this computation is accomplished in a two-stage approach: first, we use the gradient-projection method to identify variables that will be fixed at their bounds; then the quadratic model of the objective function is further reduced with respect to the free variables by using a conjugate-gradient algorithm. This iterative method is terminated at the first $s$ for which

$$\|(\nabla m_k(x_k + s))_{\text{free}}\|_\infty \leq \min\left[0.1, \max(\sqrt{\varepsilon_M}, \|\ \bar{g}(x_k)\|_\infty)\right]\ \ \|\bar{g}(x_k)\|_\infty,$$

where $(\nabla m_k(x_k + s))_{\text{free}}$ denotes the restricted gradient of the quadratic model with respect to the remaining free variables[(2)] at the beginning of the conjugate-gradient iteration and $\varepsilon_M$ is the machine precision.
Note that, if the step is not restricted to the trust region, we find the GCP without the trust-region bounds but with the real bounds of the problem and then we run the conjugate-gradient algorithm (again without the trust region). Practically, when the step is unrestricted, the algorithm is run with a huge trust-region radius.

If negative curvature is discovered for the model on an unrestricted step or if the computed trial point is unsuccessful, then the step should be recomputed with a (smaller) restriction on its length. In this case, if the distance from the current point to the previously computed generalized Cauchy point lies within the new box defined by the problem bounds and the trust-region radius, the GCP may be reused.

[(2)]Recall that the remaining free variables are those which are not fixed at the generalized Cauchy point.

As in the chapter devoted to the numerical assessment of FILTRUNC (Chapter 5), we have tested two particular variants of our code FILTBOUND. The first one, named *filter*, is the algorithm as described in Section 7.2.3, where exact first and second derivatives are used. For this variant, we have considered unsigned filter entries, that is, we use the values of the $\bar{g}(x_k)$ themselves, instead of their absolute values, in the filter test acceptance mechanism (7.12). We will present in Section 8.3.3 results for a variant with signed filter entries.

Based on practical experience presented in Gould and Toint [76], we also impose that

$$\|s_k\|_\infty \leq 1000\ \Delta_k$$

at all iterations following the first one at which a restricted step is taken. Moreover, dominated filter points are always removed from the filter in our tests (see Section 5.3 for more explanations).

The second algorithmic variant is the *pure trust-region* one, that is the same algorithm with the exception that trial points are never acceptable for the filter and the flag `RESTRICT` is always set. This variant is therefore analogous to a classical trust-region method.

Finally, every run of the algorithm is terminated if the infinity norm of the projected gradient falls below some tolerance, *i.e*, if

$$\|\bar{g}(x_k)\|_\infty \leq 10^{-6}, \tag{8.1}$$

and the flag `NONCONVEX` is unset. As discussed at the end of Section 5.3, it could happen that the filter variant stops at a point where the norm of the projected gradient is sufficiently small but with undetected negative curvature. However, we believe that, in practice, the likelihood of this situation is very small.

## 8.3 Performance and comparisons

This section is intended to present details of the numerical results obtained with the implementation of our algorithm for bound-constrained optimization. We will present a comparison of some algorithmic variants of our code and also a comparison with LANCELOT-B. All numerical results presented in this section will be compared by means of performance profiles (see Section 5.2) with the number of iterations, the number of conjugate-gradient iterations or the total CPU time as performance measures.

### 8.3.1 Filter versus pure trust-region variants

We first compare the two main algorithmic variants of FILTBOUND, namely the *filter* and the *pure trust-region* ones, both described in Section 8.2.

In order to produce the performance profiles given in this section, we have excluded four problems from the 107 box-constrained problems. We have first removed HS25 from the test set because the starting point supplied in the SIF file is already a stationary point. Three other problems have also been taken away because both variants did not report the same final objective function value for them. For problem MAXLIKA, the filter variant converges in $8$ iterations towards a point $x^*$ at which the objective function equals to $1149.3$ while the pure trust-region one stopped at a point $x^*$ satisfying $f(x^*) = 1136.3$ in 29 iterations. The other two problems are : PALMER3 that stopped at $f(x^*) = 2417$ for the filter variant and at $f(x^*) = 2266$ for the pure trust-region one, and PALMER4 that stopped at $f(x^*) = 2424$ for the filter variant and at $f(x^*) = 2285.4$ for the pure trust-region one[(3)]. The fact that filter and pure trust-region variants stop at different solutions is probably due to ill-conditioning.

We first examine the reliability of both variants. The filter variant is just marginally more robust than the pure trust-region one. Indeed, on the 107 problems, the filter variant successfully solve 101 problems and the pure trust-region one 100. Having excluded the three above-mentioned problems, both variants report the same final objective function value for problems where they both succeed. Appendix B presents, for the filter and the pure trust-region variants of FILTBOUND, the details of the runs for problems listed in Table 8.3, in particular the number of iterations and conjugate-gradient iterations, the CPU time and the final objective function value. Both variants fail on BIGGSB1, PALMER5A, PALMER7A, QRTQUAD and SCOND1LS because the maximum number of iterations has been reached before convergence is declared. The filter variant also fails, for the same reason, on MINSURFO, and the pure trust-region algorithm stalls on PALMER5B and PALMER5E. Furthermore, the pure trust-region variant is unable, for some problems, to reduce the infinity norm of the projected gradient sufficiently to meet the stopping criterion (8.1) even though the objective function value obtained is very close to the problem solution. These problems are : EXPLIN that stopped with $\|\bar{g}(x_k)\|_\infty = 1.686$E-03; EXPQUAD that stopped with $\|\bar{g}(x_k)\|_\infty = 1.453$E-02 and PALMER1A that stopped with $\|\bar{g}(x_k)\|_\infty = 3.035$E-06. However, these occurrences have been counted as successful in the performance analysis presented below.

Figures 8.1, 8.2, and 8.3 illustrate the performance profiles (for both variants) for the number of iterations, the total CPU time, and the number of conjugate-gradient iterations, respectively.

[(3)] For these two problems, the filter variant stops at the same solution than LANCELOT-B.

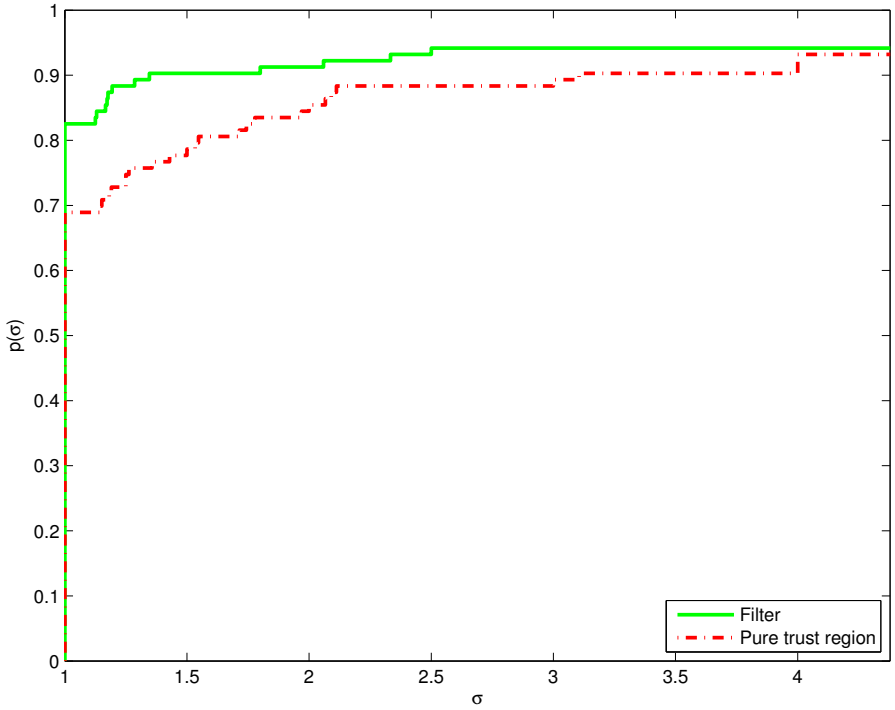


Figure 8.1: Iteration performance profile

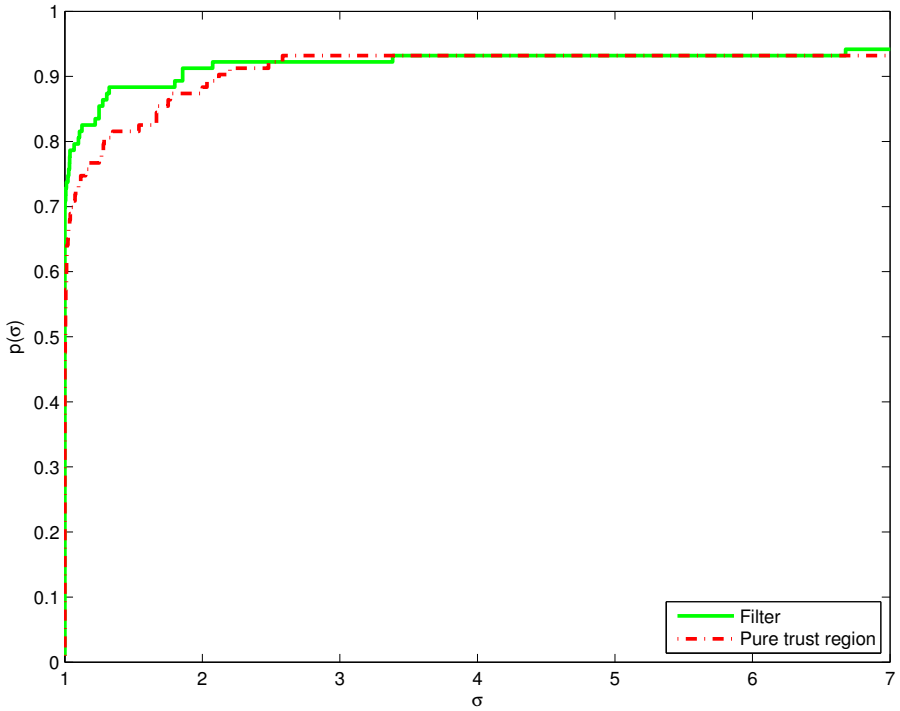


Figure 8.2: CPU performance profile

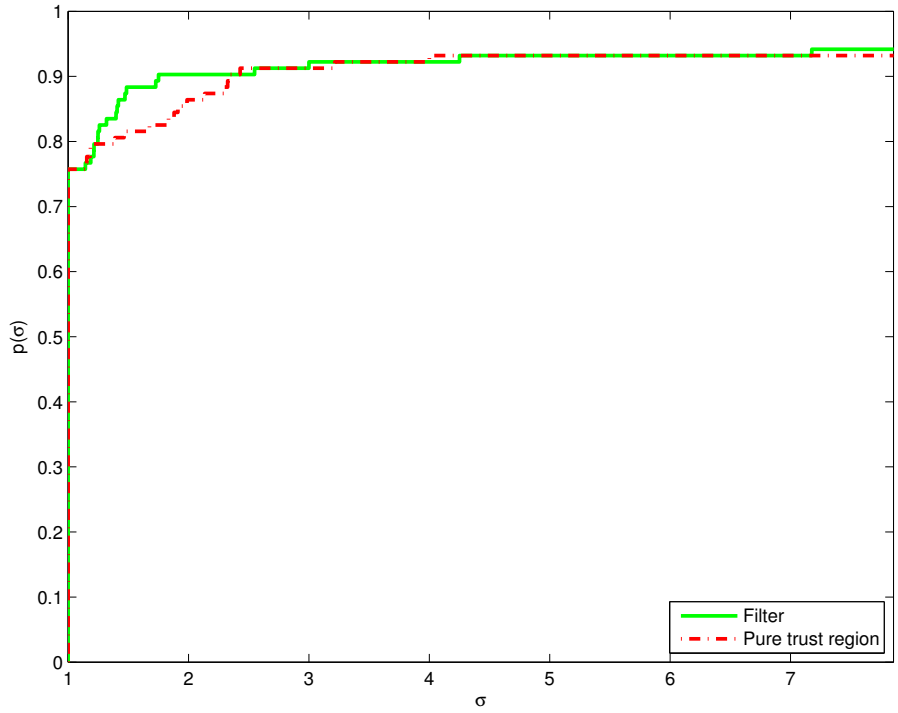


Figure 8.3: CG iteration performance profile

It can be observed in Figure 8.1 that the filter variant is the best, in terms of number of iterations, on 83% of the problems, while the pure trust-region one is the best in 69% of the cases. Although the gain in iteration count is substantial, it is less impressive than the improvement with respect to the pure trust-region method obtained for the unconstrained case (see Figure 5.1). For the unconstrained case, the benefit due to the filter mechanism is considerable in iteration count and significant but smaller in terms of conjugate-gradient iterations. The gains in terms of number of conjugate-gradient iterations and CPU time are slight here because the improvement is already smaller in terms of iterations. Filter variant is just as efficient as the pure trust-region one if we consider the CG iterations as performance measure. And the latter is just marginally more expensive in computing time than the filter variant. The high number of conjugate-gradient iterations obtained for some problems by the filter algorithm is notably due to the fact that, if a negative curvature is detected for the model during the step computation, either in the algorithm for finding the GCP or in the conjugate-gradient process, and the step is not restricted, we have to recompute a step within the intersection of the trust region and the simple bounds.

The number of filter entries for problems the filter variant can solve is distributed as indicated in Table 8.4.

| # of filter entries | # of problems |
|---|---|
| $0 \leq n_f \leq 5$ | 76 |
| $6 \leq n_f \leq 10$ | 12 |
| $11 \leq n_f \leq 50$ | 13 |

Table 8.4: The distribution of the number of filter entries

The maximum number of filter entries is moderate and never exceeds 50. As in the unconstrained case, we did not observe any relation between the filter size and the problem dimension. Indeed, successfully solved problems which introduce the most entries in the filter are : PALMER5B (30 entries) and PALMER5E (50 entries), which have, respectively, only 9 and 8 variables and EXPQUAD (31 entries), whose dimension is 1200. Furthermore, the pure trust-region variant could not solve the first two problems within the prescribed iteration limitations. It should be observed that, for the majority of problems where the filter variant fails, the algorithm puts a large number of entries in the filter. However, for those problems[(4)], the pure trust-region variant also fails. In Section 5.4.4, we have tested to limit the number of filter entries to 50 and to resort to a pure trust-region method if this number is reached. Here, as the size of the filter never exceeds 50, this algorithmic variant has no sense. We have tried the same idea with a limitation of 20 entries, but the results obtained by this variant are slightly worse than those obtained with the default filter variant.

We now detail the results (given in Table 8.5) of a problem for which the pure trust-region variant requires fewer iterations and conjugate-gradient iterations than the filter one; this problem is HS5, which is a nonlinear problem in two dimensions with its variables bounded from below and above.

| | filter | pure TR |
|---|---|---|
| # iterations | 10 | 4 |
| # CG iterations | 12 | 4 |

Table 8.5: Filter and pure trust-region runs for problem HS5

[(4)]Except for MINSURFO.

The level curves of the objective function of problem `HS5` and its bounds are displayed in Figure 8.4.

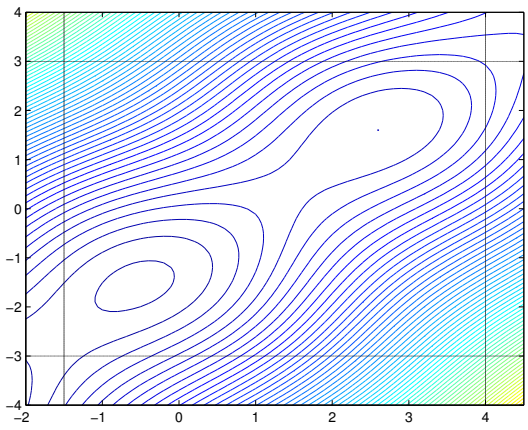


Figure 8.4: The contour lines and the box defined by the simple bounds for problem `HS5`

The plots of Figure 8.5 show the contour lines of the model around the initial point, $x_0$. The figure on the right also displays the trust region defined in $\ell_\infty$-norm around $x_0$. It can be seen that the first iterate of the filter variant moves until it reaches the boundary of the box defined by the constraints while the iterate generated by the pure trust-region algorithm is stopped by the frontier of the trust region. Furthermore, the iterate produced by the filter algorithm enters a nonconvex region.

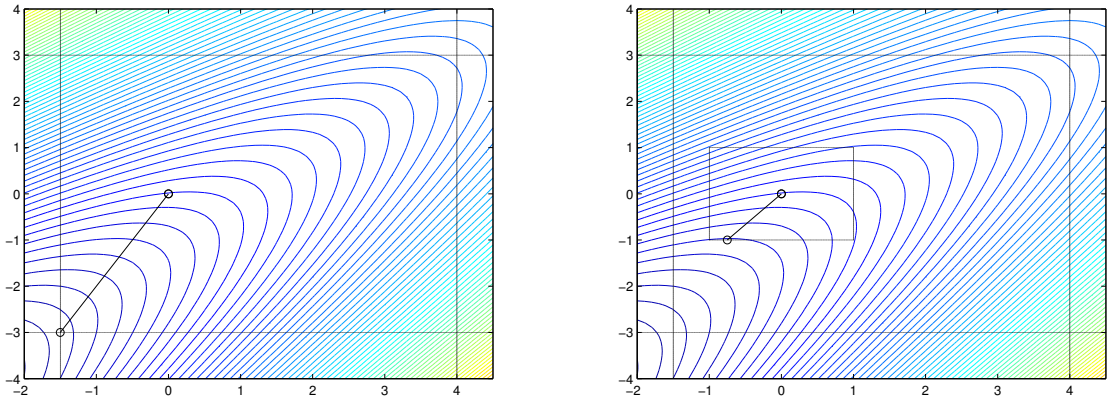


Figure 8.5: The model around the initial point and the first iterate for the filter variant (left) and the pure trust-region one (right)

The sequence of iterates is illustrated in Figure 8.6.

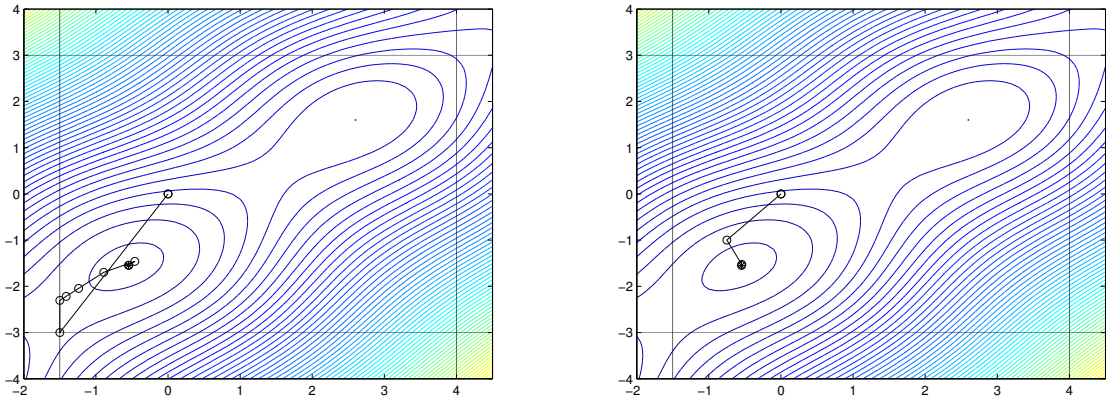

Figure 8.6: The sequence of iterates for both variants (filter on the left, pure trust region on the right)

For the iterations of the filter algorithm where negative curvature is detected in the resolution of the subproblem, we now resort to a traditional trust-region method. The following steps are then restricted to the trust region and it takes few iterations for the filter variant to converge towards the local minimum (see the picture on the left in Figure 8.6).

### 8.3.2 Comparison with LANCELOT-B

In order to assess the numerical performance of FILTBOUND, we have also compared it against LANCELOT-B (see Section 5.4.3 for a short description of this software or Conn et al. [33] for more details). Note that the computation of the trial step in LANCELOT-B and in our implementation is very similar as both of them first compute the generalized Cauchy point and then apply a conjugate-gradient method to further reduce the quadratic model $m_k$. Here we have used LANCELOT-B unpreconditioned, with exact first and second derivatives and with the initial trust-region radius equals to one. In order to be comparable with FILTBOUND for which the stopping criterion is given in (8.1), the accuracy on the projected gradient in LANCELOT-B is set to $10^{-6}$. Apart from this, we have used the default options. Note that, as previously said, LANCELOT-B is a non-monotone trust-region algorithm while our pure trust-region variant is monotone.

LANCELOT-B successfully solves 99 out of 107 problems. So it is marginally less reliable than the filter and the pure trust-region variants. Like the filter variant, LANCELOT-B does not solve BIGGSB1, PALMER5A, PALMER7A, QRTQUAD and SCOND1LS because the maximum allowed number of iterations is reached. It also fails on CHENHARK, PALMER5B and PALMER5E. Again, we have excluded MAXLIKA, PALMER3 and PALMER4 because all variants do not report the same final solution. Figures 8.7, 8.8, and 8.9 show the performance for both variants and LANCELOT-B for iteration count, CPU time, and total amount of conjugate-gradient iterations, respectively.

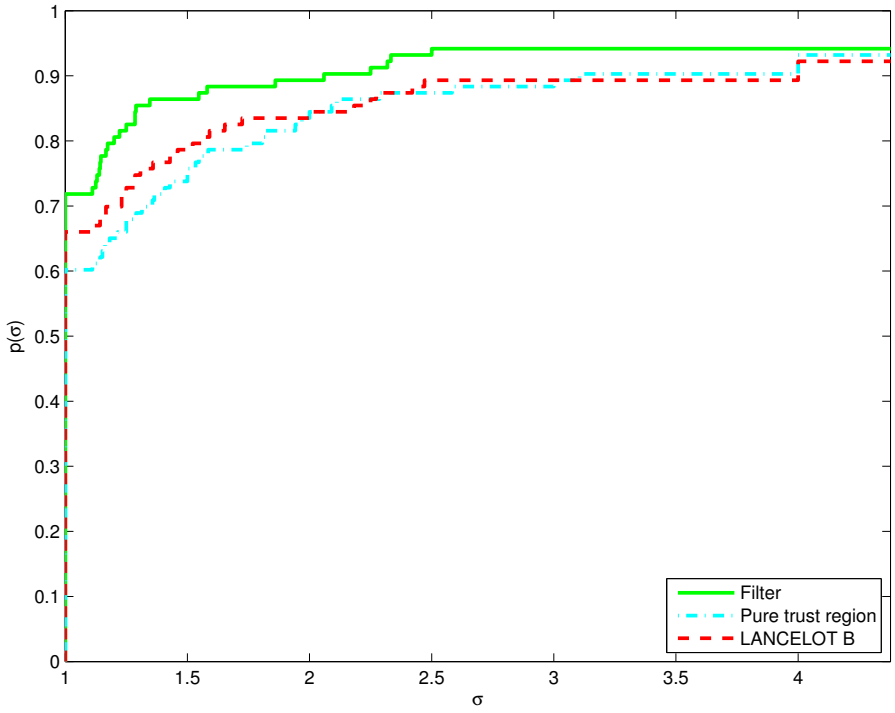


Figure 8.7: Iteration performance profile for both variants and LANCELOT-B

The LANCELOT-B solver appears to be slightly inferior to the new filter algorithm in terms of number of iterations. Indeed, if we consider iteration count as performance criterion, the filter variant is the best solver for 71.8% of the problems, while LANCELOT-B is the best in 66% of the cases. Nevertheless, this latter is more efficient in terms of conjugate-gradient iterations. In order to have a better overview of the comparison between our two variants and LANCELOT-B, we should analyse the plot where the performance measurement is the CPU time. In Figure 8.8, it can be seen that the filter variant is faster than LANCELOT-B in 58% of

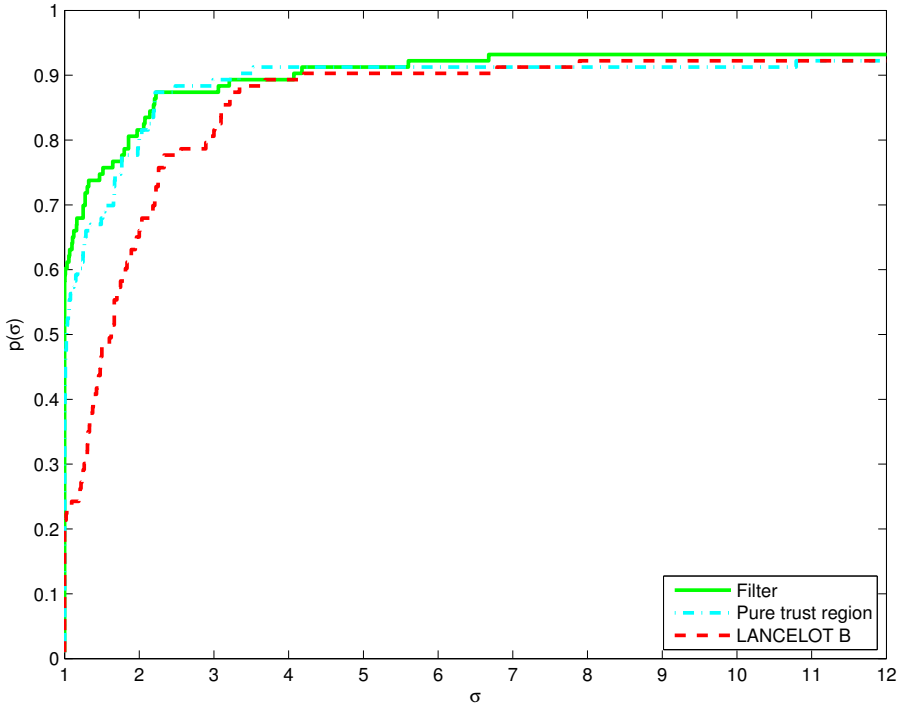


Figure 8.8: CPU performance profile for both variants and LANCELOT-B

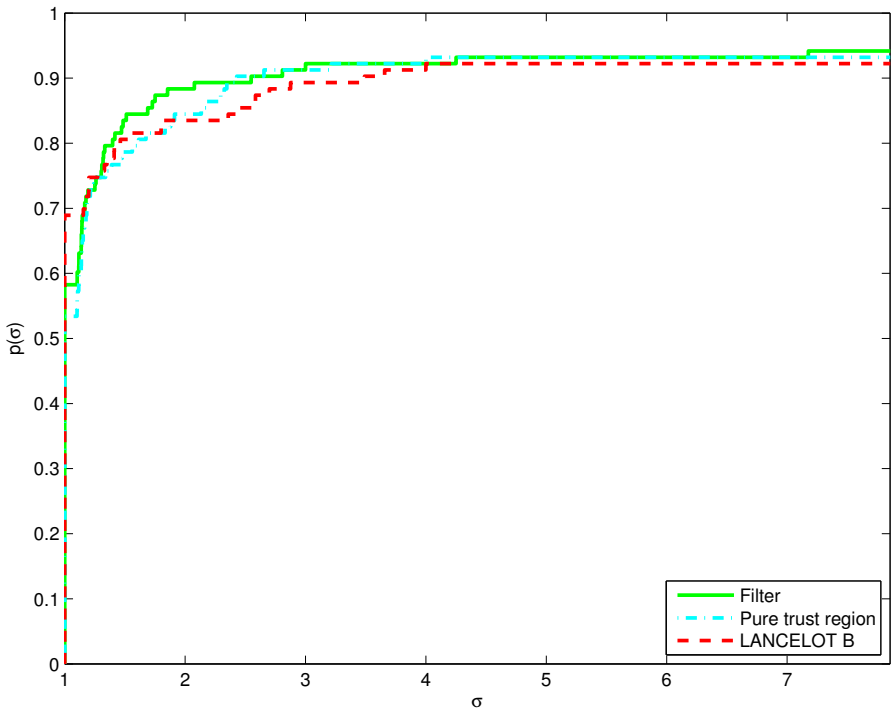


Figure 8.9: CG iteration performance profile for both variants and LANCELOT-B

the problems whereas LANCELOT-B is the quickest in 24% of the cases.

Although the numerical results are not as significant as for our filter-trust-region algorithm in the unconstrained case (see performance profiles of Section 5.4.3), we obtain interesting results.

In Figure 8.10, 8.11 and 8.12, we present some plots depicting the evolution of the objective function value for both variants and LANCELOT-B. As it was the case for similar graphs in Chapter 5, it can be observed the ample oscillations in objective function value for the filter variant. For LANCELOT-B, we can also detect a non-monotone behaviour in function value but the swings are generally smaller.

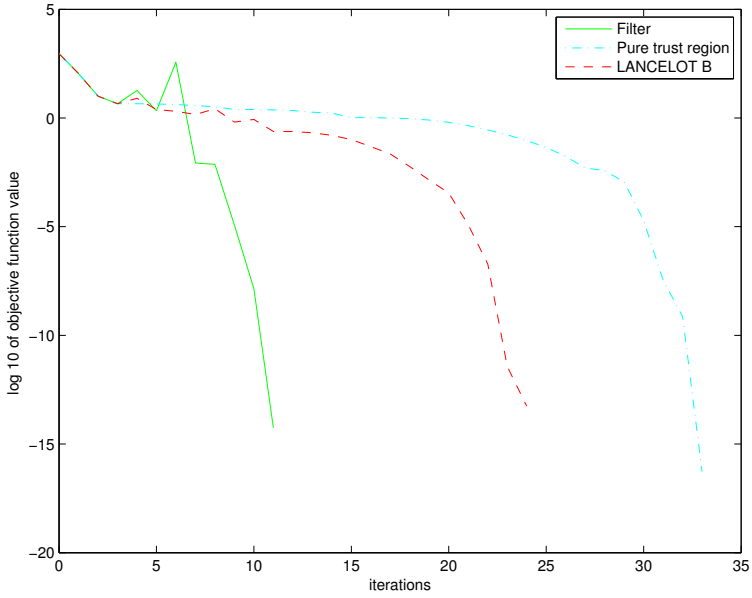


Figure 8.10: The objective function value as a function of the iteration progress on the HS1 problem for both variants and LANCELOT-B. The filter variant oscillates the most and converges first, followed by the moderately non-monotone LANCELOT-B, itself followed by the monotone pure trust-region variant.

But again, this highly non-monotone behaviour sometimes results in poor convergence as for problem PSPDOC, for which the filter variant requires 14 iterations to converge while the pure trust-region one and LANCELOT-B, which coincide in Figure 8.13, require only 6 iterations. The first iteration generated by applying our filter algorithm on PSPDOC is acceptable because of the filter.

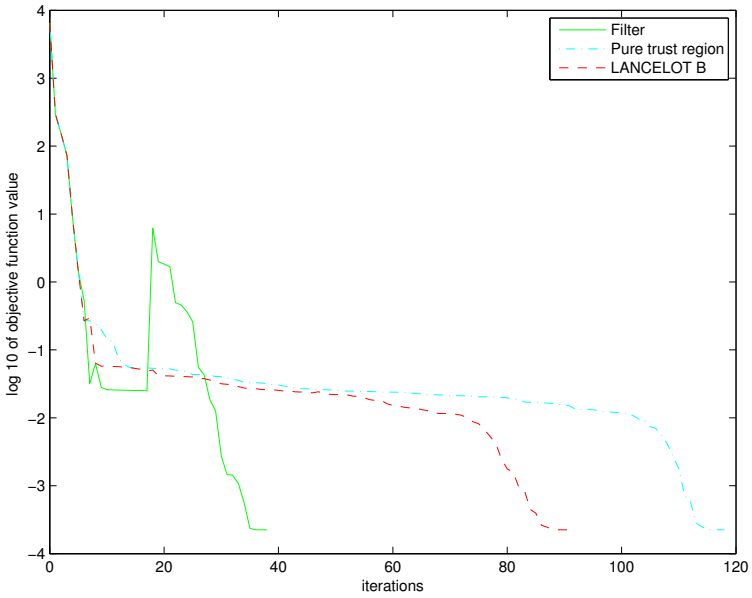


Figure 8.11: The objective function value as a function of the iteration progress on the PALMER6E problem for both variants and LANCELOT-B

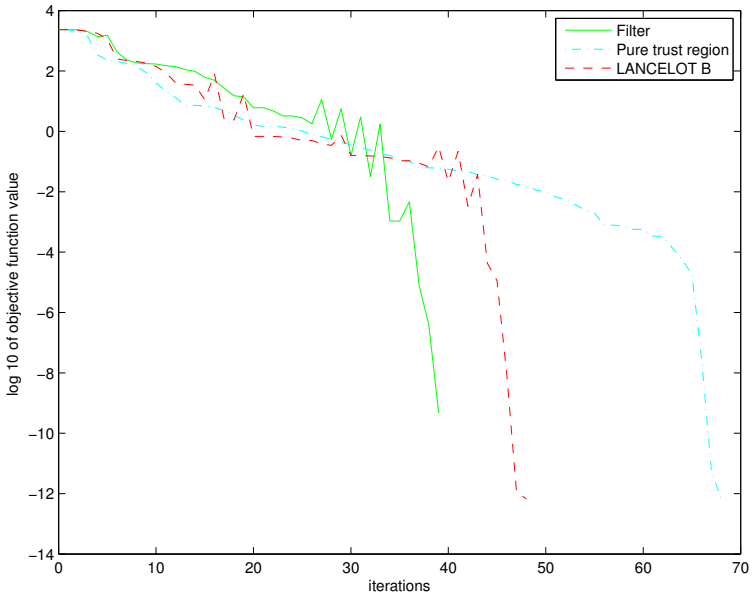


Figure 8.12: The objective function value as a function of the iteration progress on the YFIT problem for both variants and LANCELOT-B

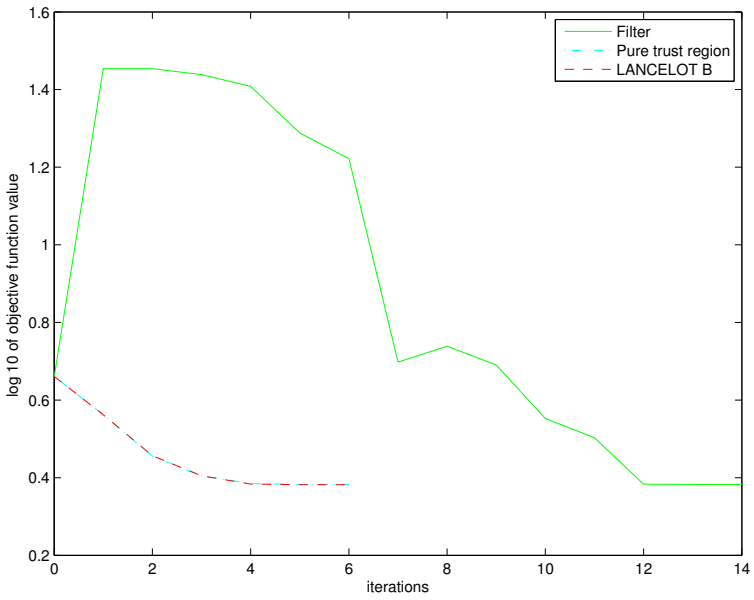


Figure 8.13: The objective function value as a function of the iteration progress on the PSPDOC problem for both variants and LANCELOT-B

### 8.3.3 Signed filter entries

The last section of the performance analysis is intended to compare the default filter variant of FILTBOUND with a variant using signed filter entries, called *signed* variant in contrast with the *unsigned* default one. So we now use the absolute values of the $\bar{g}(x_k)$ in the filter acceptance test.

The efficiency and robustness of the signed and unsigned variants are illustrated in Figure 8.14. It can be shown that the signed variant is a little less reliable[(5)] (100/107 versus 101/107) and also a little bit less efficient than the unsigned one. However, we can notice in Figure 8.15 that the gain obtained by the unsigned variant results in a higher number of inclusions in the filter.

## 8.4 Conclusion

The preliminary numerical results obtained on the set of box-constrained problems from the CUTEr collection show a general good behaviour of our filter-trust-region algorithm. Although the numerical improvement with respect to the pure trust-region method is less significant than for the unconstrained case, we still believe that the resulting method is of interest, and that it is

[(5)]The signed variant failed to solve PALMER5E.

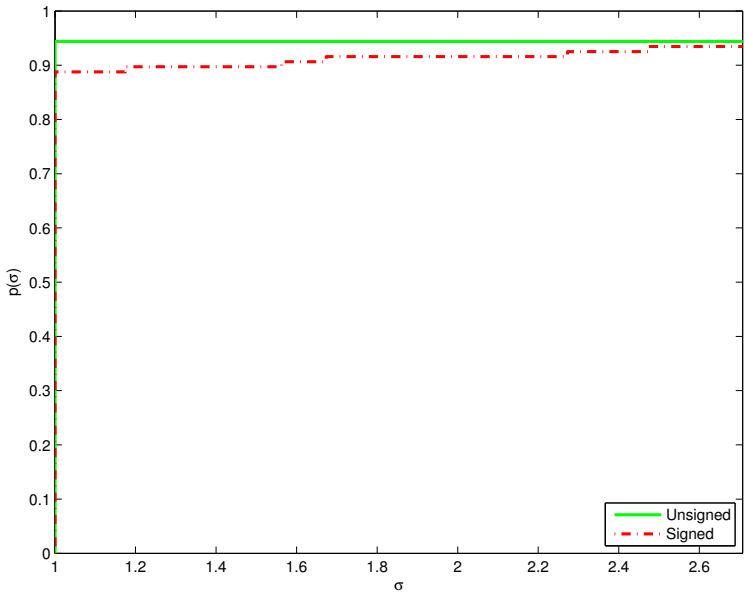


Figure 8.14: Iteration performance profile for the filter variants using unsigned and signed filter entries

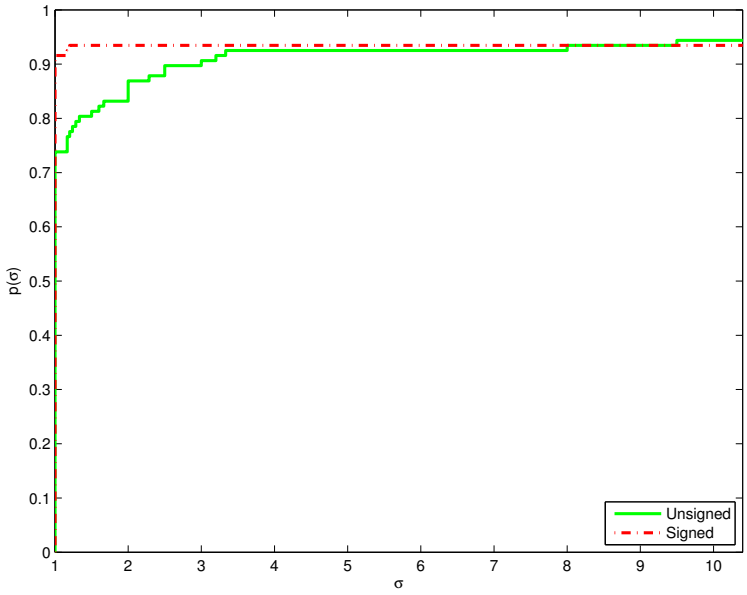


Figure 8.15: Filter size performance profile for the filter variants using unsigned and signed filter entries

potentially useful. This study indicates that, in terms of computing time, the new algorithm is very competitive with LANCELOT-B on many problems. Finally, we believe that it should be interesting to introduce our filter mechanism in LANCELOT-B.

# Conclusions and further research perspectives

Nonlinear optimization is a highly active research field. The purpose of the research work described in this thesis was the design and implementation of algorithms for solving two classes of mathematical programs, namely unconstrained and bound-constrained nonlinear optimization problems. For both of them, we have developed algorithms using a filter mechanism within a trust-region scheme.

In the next two paragraphs, both kinds of mathematical problems we have been investigating are reviewed and we summarize our results. At the same time, we also suggest some possible directions for future research.

## Filter-trust-region method for unconstrained optimization

In the first part of this thesis, we have presented and analyzed a novel algorithm for solving unconstrained optimization problems. The method uses a trust-region scheme combined with a multidimensional filter technique whose aim is to be more permissive in the acceptance of trial points than classical trust-region methods. One of the advantages of our filter-based algorithm is the possibility of accepting steps whose norm exceeds the trust-region boundary, *i.e.* such that

$$\|s_k\| > \Delta_k,$$

without affecting the convergence properties of the method. Another important feature of our approach is the non-monotone behaviour in objective function values. Our algorithm was shown to produce, under mild assumptions, at least a first-order critical point, irrespective of the chosen starting point. Under additional conditions, it has also been proved that convergence of the complete sequence of iterates generated by our algorithm can only occur to a second-order critical point. The numerical tests performed on the CUTEr collection presented in Sections 5.4.1 to 5.4.3 show that our new algorithm for unconstrained optimization is very competitive with traditional trust-region algorithms. While the numerical results are very encouraging, there is

always room for improvement. Continued testing and better adjustments of parameter values should be necessary.
An idea of future work should be to consider the objective function as an entry of the filter as in most filter methods for constrained optimization.

In Chapter 6, we have presented a numerical investigation of the influence of approximate derivatives on the robustness and efficiency of our filter-trust-region method. This study is performed on small-scale problems of the CUTEr test set. In general, we would recommend our algorithm with exact derivatives when the latter are available. However, if second-order information is not available or too expensive, our algorithm allows the approximation of the Hessian either by means of finite differences or secant updates, namely BFGS and SR1. The comparison of both secant approximations in Section 6.2.2 indicates that, mostly, the variant of our algorithm using SR1 updates has better performance than a variant with BFGS techniques, but, for some problems, it is the other way round. The best updating scheme is therefore problem-dependent, which underlines the importance of flexibility of an optimization code. Typically variants with second-order derivatives updated by secant formulae require more iterations to converge than the version with exact Hessian. However, since it is only computing gradients rather than Hessians, this approach may be more efficient in some cases. Our implementation stores a dense Hessian approximation so it is only recommended for small to medium problems. In order to provide a high performance solver for large-scale problems in the case where derivatives are unavailable, we could conceive the use of a limited-memory quasi-Newton technique as in the code L-BFGS-B.

## Filter-trust-region method for bound-constrained optimization

The second main contribution builds up on the first. It was shown how to extend our filter-trust-region algorithm to bound-constrained optimization problems. A new algorithm, making use of three tools of nonlinear programming, namely filter techniques, trust regions and gradient-projection methods, has been proposed. Our algorithm has been shown, under standard assumptions, to produce at least a first-order critical point, irrespective of the initial point. However, the optimal active-set identification and the second-order convergence analysis remains to be done.
The proposed algorithm has been tested on a large collection of test problems and compared with LANCELOT-B. Given the variety and difficulty of nonlinear optimization problems, it is unlikely that a unique best algorithm will emerge. Although the numerical results reported with our filter-trust-region algorithm for bound-constrained optimization is less impressive than for the unconstrained case, we still believe that the resulting method is of interest, and that it is

potentially useful. We also guess that some improvements may be obtained to enhance the performance of this code. We should, in particular, modify the solution method for the trust-region subproblem to try to decrease the number of conjugate-gradient iterations and CPU time. This number can possibly be reduced by using appropriate preconditioners. An inexact generalized Cauchy point may also be computed instead of an exact one. Furthermore, in our computation of the trial step, when the conjugate-gradient iterates attempt to go out the feasible box, we fix variables to their bounds and restart the conjugate-gradient process. However, there exist more sophisticated algorithms allowing infeasible conjugate-gradient iterates. During the subproblem computation, the infeasible iterates are, at regular intervals, projected back onto the feasible region. A similar technique could be tried in our implementation in order to improve the performances.

A further work should be the inclusion of a multidimensional filter mechanism similar to those presented in this dissertation in the LANCELOT-B software.

Another potential extension of our filter-trust-region method should be its use for nonlinear complementarity problems (NCP), that is problems of the following form

$$\langle F(x), x\rangle \quad F(x) \geq 0 \quad x \geq 0,$$

where $F : \mathbb{R}^n \rightarrow \mathbb{R}^n$. This kind of problems may be treated by reformulating the problem as an unconstrained minimization problem of the form

$$\min_{x\in\mathbb{R}^n} \tfrac{1}{2}\|\Phi(x)\|^2,$$

where $(\Phi(x))_i \stackrel{\text{def}}{=} \varphi(x_i, F_i(x)) \ \forall i$ and $\varphi(\cdot,\cdot)$ is an NCP function, like, for instance, the Fisher-Burmeister function. We then should apply a modified version of our multidimensional filter algorithm to this unconstrained reformulation. However, our implementation as it stands requires the problem to be written in SIF and, actually, there is no NCP problems written in SIF in the CUTEr collection. So either we would have to translate existing nonlinear complementarity problems in SIF in order to obtain a collection of problems to test the algorithm, or we would have to adapt our software to allow problems written, for example, in AMPL.

Finally, it is our hope that the work described in this thesis can be useful in future algorithmic developments for nonlinear optimization.

# Summary of contributions

Our contributions are the design, the theoretical study and the implementation of algorithms for solving two classes of mathematical programs, namely nonlinear unconstrained optimization problems (see Part I) and nonlinear bound-constrained optimization problems (see Part II). We summarize our contributions below.

- **Unconstrained optimization**
  - ∗ The filter-trust-region algorithm for solving unconstrained nonlinear mathematical programs (Section 4.1) as well as its convergence analysis (Section 4.2) and the numerical results (Chapter 5) obtained with our code FILTRUNC have been first presented in [72]. This algorithm represents one of the first applications of filter techniques to general unconstrained optimization.
  - ∗ The investigation on the use of approximate derivatives in our filter-trust-region algorithm (Chapter 6) has been discussed in [110].
- **Bound-constrained optimization**

  The design of the filter-trust-region algorithm for solving bound-constrained optimization problems (Section 7.2), its convergence properties (Section 7.3) and the numerical experience presented in Chapter 8 are the subject of the paper [111].

From a practical point of view, we have implemented our algorithms in Fortran 90, tested them on large set of test problems and compared their performances with LANCELOT-B.

Note that a comparison of some existing filter-based methods can be found in our Master's thesis [109], published in 2002.

# Bibliography


[1] M. Andretta, E. G. Birgin, and J. M. Martínez. Practical active-set euclidian trust-region method with spectral projected gradients for bound-constrained optimization. *Optimization*, 54(3):305–325, 2005.

[2] C. Audet and J. E. Dennis. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13(3):889–903, 2003.

[3] C. Audet and J. E. Dennis. A pattern search filter method for nonlinear programming without derivatives. *SIAM Journal on Optimization*, 14(4):980–1010, 2004.

[4] M. S. Bazaraa, H. Sherali, and C. Shetty. *Nonlinear Programming: Theory and Applications*. J. Wiley and Sons, Chichester, England, 1993.

[5] E. M. L. Beale. Numerical methods. In J. Abadie, editor, *Nonlinear programming*, pages 135–205. North Holland, Amsterdam, The Netherlands, 1967.

[6] H. Y. Benson, R. J. Vanderbei, and D. F. Shanno. Interior-Point Methods for Nonconvex Nonlinear Programming: Filter Methods and Merit Functions. *Computational Optimization and Applications*, 23:257–272, 2002.

[7] D. P. Bertsekas. On the Goldstein-Levitin-Polyak gradient projection method. *IEEE Transactions on Automatic Control*, 21:174–184, 1976.

[8] D. P. Bertsekas. *Nonlinear Programming*. Athena Scientific, Belmont, Massachussetts, USA, 1995.

[9] M. Bierlaire. A robust algorithm for the simultaneous estimation of hierarchical logit models. TRG Report 95/3, Transportation Research Group, Department of Mathematics, University of Namur, Namur, Belgium, 1995.

[10] J. R. Birge and F. Louveaux. *Introduction to Stochastic Programming*. Springer Verlag, Heidelberg, Berlin, New York, 1997.

[11] P. T. Boggs and J. E. Dennis. A stability analysis for perturbed nonlinear iterative methods. *Mathematics of Computation*, 30(134):199–215, 1976.

[12] P. T. Boggs and J. W. Tolle. Sequential quadratic programming. *Acta Numerica*, 4:1–51, 1995.

[13] I. Bongartz, A. R. Conn, N. I. M. Gould, and Ph. L. Toint. CUTE: Constrained and Unconstrained Testing Environment. *ACM Transactions on Mathematical Software*, 21(1):123–160, 1995.

[14] J. F. Bonnans, J. Ch. Gilbert, C. Lemaréchal, and C. A. Sagastizábal. *Numerical Optimization : Theoretical and Practical Aspects*. Springer Verlag, Heidelberg, Berlin, New York, 2002.

[15] C. G. Broyden. The convergence of a class of double-rank minimization algorithms. *Journal of the Institute of Mathematics and its Applications*, 6:76–90, 1970.

[16] C. G. Broyden, J. E. Dennis, and J. J. Moré. On the local and superlinear convergence of quasi-Newton methods. *Journal of the Institute of Mathematics and its Applications*, 12:233–246, 1973.

[17] R. H. Byrd, N. I. M. Gould, J. Nocedal, and R. A. Waltz. An algorithm for nonlinear optimization using linear programming and equality constrained subproblems. *Mathematical Programming, Series B*, 100(1):27–48, 2004.

[18] R. H. Byrd, H. F. Khalfan, and R. B. Schnabel. Analysis of a symmetric rank-one trust region method. *SIAM Journal on Optimization*, 6(4):1025–1039, 1996.

[19] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu. A limited memory algorithm for bound constrained optimization. *SIAM Journal on Scientific Computing*, 16(5):1190–1208, 1995.

[20] R. G. Carter. On the global convergence of trust region methods using inexact gradient information. *SIAM Journal on Numerical Analysis*, 28(1):251–265, 1991.

[21] C. Charalambous. A lower bound for the controlling parameter of the exact penalty functions. *Mathematical Programming*, 15(3):278–290, 1978.

[22] Ch. M. Chin and R. Fletcher. Convergence properties of SLP-filter algorithms that takes EQP steps. *Mathematical Programming, Series A*, 96(1):161–177, 2003.

[23] V. Chvátal. *Linear Programming*. W. H. Freeman and Company, New York and San Francisco, 1983.

[24] T. F. Coleman and A. R. Conn. Nonlinear programming via an exact penalty function method : Asymptotic analysis. *Mathematical Programming*, 24(1):123–136, 1982.

[25] T. F. Coleman and A. R. Conn. Nonlinear programming via an exact penalty function method: Global analysis. *Mathematical Programming*, 24(1):137–161, 1982.

[26] B. Colson. *Trust-Region Algorithms for Derivative-Free Optimization and Nonlinear Bilevel Programming*. PhD thesis, Department of Mathematics, University of Namur, Namur, Belgium, 2003.

[27] A. R. Conn. Constrained optimization via a nondifferentiable penalty function. *SIAM Journal on Numerical Analysis*, 10(4):760–779, 1973.

[28] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Global convergence of a class of trust region algorithms for optimization with simple bounds. *SIAM Journal on Numerical Analysis*, 25(182):433–460, 1988. See also same journal 26:764–767, 1989.

[29] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Testing a class of methods for solving minimization problems with simple bounds on the variables. *Mathematics of Computation*, 50:399–430, 1988.

[30] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Convergence of quasi-Newton matrices generated by the symmetric rank one update. *Mathematical Programming*, 50(2):177–196, 1991.

[31] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. A globally convergent Lagrangian barrier algorithm for optimization with general inequality constraints and simple bounds. Technical Report 92/07, Department of Mathematics, University of Namur, Namur, Belgium, 1992.

[32] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. Intensive numerical tests with LANCELOT (Release A): the complete results. Technical Report 92/15, Department of Mathematics, University of Namur, Namur, Belgium, 1992. Also issued as Research Report RC 18750, IBM T.J. Watson Center, Yorktown Heights, USA, and as Research Report 92-069, RAL, Chilton, Oxfordshire, England.

[33] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. LANCELOT*: a Fortran package for large-scale nonlinear optimization (Release A)*. Number 17 in Springer Series in Computational Mathematics. Springer Verlag, Heidelberg, Berlin, New York, 1992.

[34] A. R. Conn, N. I. M. Gould, and Ph. L. Toint. *Trust-Region Methods*. Number 01 in MPS-SIAM Series on Optimization. SIAM, Philadelphia, USA, 2000.

[35] A. R. Conn, K. Scheinberg, and Ph. L. Toint. On the convergence of derivative-free methods for unconstrained optimization. In A. Iserles and M. Buhmann, editors, *Approximation Theory and Optimization: Tributes to M. J. D. Powell*, pages 83–108, Cambridge, England, 1997. Cambridge University Press.

[36] A. R. Conn, K. Scheinberg, and Ph. L. Toint. Recent progress in unconstrained nonlinear optimization without derivatives. *Mathematical Programming, Series B*, 79(3):397–414, 1997.

[37] G. B. Dantzig. *Linear Programming and Extensions*. Princeton Uinversity Press, Princeton, USA, 1963.

[38] W. C. Davidon. Variable metric method for minimization. Report ANL-5990(Rev.), Argonne National Laboratory, Research and Development, 1959.

[39] V. F. Dem'yanov and L.V. Vasil'ev. *Nondifferentiable Optimization*. Springer Verlag, New York, 1985.

[40] J. E. Dennis and J. J. Moré. Quasi-Newton methods, motivation and theory. *SIAM Review*, 19:46–89, 1977.

[41] J. E. Dennis and R. B. Schnabel. *Numerical Methods for Unconstrained Optimization and Nonlinear Equations*. Prentice-Hall, Englewood Cliffs, New Jersey, USA, 1983. Reprinted as *Classics in Applied Mathematics 16*, SIAM, Philadelphia, USA, 1996.

[42] G. Di Pillo and A. Murli, editors. *High Performance Algorithms and Software in Nonlinear Optimization*, Dordrecht, The Netherlands, 2003. Kluwer Academic Publishers.

[43] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical Programming*, 91(2):201–213, 2002.

[44] Z. Dostál. Box constrained quadratic programming with proportioning and projections. *SIAM Journal on Optimization*, 7(3):871–887, 1997.

[45] A. V. Fiacco and G. P. McCormick. Programming under nonlinear constraints by unconstrained optimization: a primal-dual method. Technical Report RAC-TP-96, Research Analysis Corporation, McLean, Virginia, USA, 1963.

[46] A. V. Fiacco and G. P. McCormick. The sequential unconstrained minimization technique for nonlinear programming: a primal-dual method. *Management Science*, 10(2):360–366, 1964.

[47] R. Fletcher. A new approach to variable metric algorithms. *Computer Journal*, 13:317–322, 1970.

[48] R. Fletcher. *Practical Methods of Optimization*. J. Wiley and Sons, Chichester, England, second edition, 1987.

[49] R. Fletcher, N. I. M. Gould, S. Leyffer, Ph. L. Toint, and A. Wächter. Global convergence of trust-region SQP-filter algorithms for nonlinear programming. *SIAM Journal on Optimization*, 13(3):635–659, 2002.

[50] R. Fletcher and S. Leyffer. User manual for filterSQP. Numerical Analysis Report NA/181, Department of Mathematics, University of Dundee, Dundee, Scotland, 1998.

[51] R. Fletcher and S. Leyffer. A bundle filter method for nonsmooth nonlinear optimization. Numerical Analysis Report NA/195, Department of Mathematics, University of Dundee, Dundee, Scotland, 1999.

[52] R. Fletcher and S. Leyffer. Nonlinear programming without a penalty function. *Mathematical Programming*, 91(2):239–269, 2002.

[53] R. Fletcher and S. Leyffer. Filter-type algorithms for solving systems of algebraic equations and inequalities. In Di Pillo and Murli [42], pages 259–278.

[54] R. Fletcher, S. Leyffer, and Ph. L. Toint. On the global convergence of a SLP-filter algorithm. Technical Report 98/13, Department of Mathematics, University of Namur, Namur, Belgium, 1998.

[55] R. Fletcher, S. Leyffer, and Ph. L. Toint. On the global convergence of a filter-SQP algorithm. *SIAM Journal on Optimization*, 13(1):44–59, 2002.

[56] R. Fletcher and M. J. D. Powell. A rapidly convergent descent method for minimization. *Computer Journal*, 6:163–168, 1963.

[57] A. Forsgren, P. E. Gill, and M. H. Wright. Interior-point methods for nonlinear optimization. *SIAM Review*, 44:525–597, 2002.

[58] R. Fourer and D. Orban. The DrAmpl meta solver for optimization. Technical report, GERAD, Montreal, Canada, 2007.

[59] P. E. Gill and W. Murray, editors. *Numerical Methods for Constrained Optimization*, London, 1974. Academic Press.

[60] P. E. Gill, W. Murray, and M. A. Saunders. SNOPT: An SQP algorithm for large-scale constrained optimization. *SIAM Journal on Optimization*, 12(4):979–1006, 2002.

[61] P. E. Gill, W. Murray, M. A. Saunders, and M. Wright. Computing forward-difference intervals for numerical optimization. *SIAM Journal on Scientific and Statistical Computing*, 4:310–321, 1983.

[62] P. E. Gill, W. Murray, and M. H. Wright. *Practical Optimization*. Academic Press, London, 1981.

[63] D. Goldfarb. A family of variable metric methods derived by variational means. *Mathematics of Computation*, 24:23–26, 1970.

[64] C. C. Gonzaga, E. Karas, and M. Vanti. A globally convergent filter method for nonlinear programming. *SIAM Journal on Optimization*, 14(3):646–669, 2003.

[65] N. I. M. Gould, S. Leyffer, and Ph. L. Toint. A multidimensional filter algorithm for nonlinear equations and nonlinear least-squares. *SIAM Journal on Optimization*, 15(1):17–38, 2005.

[66] N. I. M. Gould, S. Lucidi, M. Roma, and Ph. L. Toint. Solving the trust-region subproblem using the Lanczos method. *SIAM Journal on Optimization*, 9(2):504–525, 1999.

[67] N. I. M. Gould, D. Orban, A. Sartenaer, and Ph. L. Toint. Sensitivity of trust-region algorithms on their parameters. *4OR, Quarterly Journal of the Italian, French and Belgian OR Societies*, 3, 2005.

[68] N. I. M. Gould, D. Orban, and Ph. L. Toint. Results from a numerical evaluation of lancelot b. Technical Report 02/09, Department of Mathematics, University of Namur, Namur, Belgium, 2002.

[69] N. I. M. Gould, D. Orban, and Ph. L. Toint. CUTEr, a constrained and unconstrained testing environment, revisited. *ACM Transactions on Mathematical Software*, 29(4):373–394, 2003.

[70] N. I. M. Gould, D. Orban, and Ph. L. Toint. GALAHAD—a library of thread-safe Fortran 90 packages for large-scale nonlinear optimization. *ACM Transactions on Mathematical Software*, 29(4):353–372, 2003.

[71] N. I. M. Gould, D. Orban, and Ph. L. Toint. Numerical methods for large-scale nonlinear optimization. *Acta Numerica*, 14:299–361, 2005.

[72] N. I. M. Gould, C. Sainvitu, and Ph. L. Toint. A filter-trust-region method for unconstrained optimization. *SIAM Journal on Optimization*, 16(2):341–357, 2005.

[73] N. I. M. Gould and Ph. L. Toint. SQP methods for large-scale nonlinear programming. In M. J. D. Powell and S. Scholtes, editors, *System Modelling and Optimization, Methods, Theory and Applications*, pages 149–178, Dordrecht, The Netherlands, 2000. Kluwer Academic Publishers.

[74] N. I. M. Gould and Ph. L. Toint. Global convergence of a hybrid trust-region SQP-filter algorithm for general nonlinear programming. In E. Sachs and R. Tichatschke, editors, *System Modeling and Optimization XX*, pages 23–54, Dordrecht, The Netherlands, 2003. Kluwer Academic Publishers.

[75] N. I. M. Gould and Ph. L. Toint. Global convergence of a non-monotone trust-region filter algorithm for nonlinear programming. In W. Hager, editor, *Proceedings of the 2004 Gainesville Conference on Multilevel Optimization*, Dordrecht, The Netherlands, 2005. Kluwer Academic Publishers.

[76] N. I. M. Gould and Ph. L. Toint. FILTRANE, a Fortran 95 filter-trust-region package for solving nonlinear least-squares problems and nonlinear feasibility problems. *ACM Transactions on Mathematical Software*, 2007.

[77] A. Griewank. On automatic differentiation. In Iri and Tanabe [82], pages 83–108.

[78] A. Griewank. Computational differentiation and optimization. In J. R. Birge and K. G. Murty, editors, *Mathematical Programming: State of the Art 1994*, pages 102–131, Ann Arbor, USA, 1994. The University of Michigan.

[79] A. Griewank and G. Corliss. *Automatic Differentiation of Algorithms: Theory, Implementation and Application*. SIAM, Philadelphia, USA, 1991.

[80] S. P. Han and O. L. Mangasarian. Exact penalty functions in nonlinear programming. *Mathematical Programming*, 17(3):251–269, 1979.

[81] M. R. Hestenes and E. Stiefel. Methods of conjugate gradients for solving linear systems. *Journal of the National Bureau of Standards*, 49:409–436, 1952.

[82] M. Iri and K. Tanabe, editors. *Mathematical Programming: Recent Developments and Applications*, Dordrecht, The Netherlands, 1989. Kluwer Academic Publishers.

[83] P. Kall and S. W. Wallace. *Stochastic Programming*. J. Wiley and Sons, Chichester, England, 1994.

[84] N. Karmarkar. A new polynomial-time algorithm for linear programming. *Combinatorica*, 4:373–395, 1984.

[85] W. Karush. Minima of functions of several variables with inequalities as side conditions. Master's thesis, Department of Mathematics, University of Chicago, Illinois, USA, 1939.

[86] H. W. Kuhn and A. W. Tucker. Nonlinear programming. In J. Neyman, editor, *Proceedings of the second Berkeley symposium on mathematical statistics and probability*, California, USA, 1951. University of Berkeley Press.

[87] K. Levenberg. A method for the solution of certain problems in least squares. *Quarterly Journal on Applied Mathematics*, 2:164–168, 1944.

[88] E. S. Levitin and B. T. Polyak. Constrained minimization problems. *U.S.S.R. Comput. Math. Math. Phys.*, 6:1–50, 1966.

[89] C. Lin and J. J. Moré. Newton's method for large bound-constrained optimization problems. *SIAM Journal on Optimization*, 9(4):1100–1127, 1999.

[90] D. C. Liu and J. Nocedal. On the limited memory BFGS method for large scale optimization. *Mathematical Programming, Series B*, 45(1):503–528, 1989.

[91] D. G. Luenberger. *Linear and Nonlinear Programming*. Addison-Wesley Publishing Company, Reading, Massachusetts, USA, second edition, 1984.

[92] D. Marquardt. An algorithm for least-squares estimation of nonlinear parameters. *SIAM Journal on Applied Mathematics*, 11:431–441, 1963.

[93] K. Miettinen. *Nonlinear Multiobjective Optimization*, volume 12 of *Internationals Series in Operations Research & Management Science*. Kluwer Academic Publishers, Boston, USA, 1999.

[94] J. J. Moré. Trust regions and projected gradients. In M. Iri and K. Yajima, editors, *System Modelling and Optimization*, volume 113, pages 1–13, Heidelberg, Berlin, New York, 1988. Springer Verlag. Lecture Notes in Control and Information Sciences.

[95] J. J. Moré and D. C. Sorensen. Computing a trust region step. *SIAM Journal on Scientific and Statistical Computing*, 4(3):553–572, 1983.

[96] J. J. Moré and G. Toraldo. On the solution of large quadratic programming problems with bound constraints. *SIAM Journal on Optimization*, 1(1):93–113, 1991.

[97] B. A. Murtagh. *Advanced Linear Programming*. McGraw-Hill, New York, USA, 1981.

[98] S. G. Nash and A. Sofer. *Linear and nonlinear Programming*. McGraw-Hill, New York, USA, 1996.

[99] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. J. Wiley and Sons, Chichester, England, 1988.

[100] J. Nocedal. Updating quasi-Newton matrices with limited storage. *Mathematics of Computation*, 35:773–782, 1980.

[101] J. Nocedal and S. J. Wright. *Numerical Optimization*. Series in Operations Research. Springer Verlag, Heidelberg, Berlin, New York, 1999.

[102] J. Nocedal and Y. Yuan. Combining trust region and line search techniques. In Yuan [130], pages 153–176.

[103] E. O. Omojokun. *Trust region algorithms for optimization with nonlinear equality and inequality constraints*. PhD thesis, University of Colorado, Boulder, Colorado, USA, 1989.

[104] J. M. Ortega and W. C. Rheinboldt. *Iterative Solution of Nonlinear Equations in Several Variables*. Academic Press, London, 1970.

[105] T. Pietrzykowski. An exact potential method for constrained maxima. *SIAM Journal on Numerical Analysis*, 6(2):299–304, 1969.

[106] M. W. Trosset R. M. Lewis, V. Torczon. Direct search methods: Then and now. *Journal of Computational and Applied Mathematics*, 124:191–207, 2000.

[107] A. A. Ribeiro, E. W. Karas, and C. C. Gonzaga. Global convergence of filter methods for nonlinear programming. Technical report, Department of Mathematics, Federal University of Panamá, Curitiba, Brazil, 2006.

[108] A. Ruszczyński. *Nonlinear Optimization*. Princeton Uinversity Press, Princeton, USA, 2006.

[109] C. Sainvitu. Les méthodes de filtre en optimisation non-linéaire : une comparaison des variantes avec recherche linéaire et région de confiance. Master's thesis, Department of Mathematics, University of Namur, Namur, Belgium, 2002.

[110] C. Sainvitu. A numerical investigation of the influence of approximate derivatives on the filter-trust-region method for unconstrained optimization. Technical Report 06/03, Department of Mathematics, University of Namur, Namur, Belgium, 2006. (Submitted to *RAIRO-Recherche Opérationnelle—Operations Research*).

[111] C. Sainvitu and Ph. L. Toint. A filter-trust-region method for simple-bound constrained optimization. *Optimization Methods and Software*, (accepted for publication), 2007.

[112] R. W. H. Sargent. Reduced-gradient and projection methods for nonlinear programming. In Gill and Murray [59], pages 149–174.

[113] A. Sartenaer. Automatic determination of an initial trust region in nonlinear programming. *SIAM Journal on Scientific Computing*, 18(6):1788–1803, 1997.

[114] A. Sartenaer. Some recent developments in nonlinear optimization algorithms. In *ESAIM: Proceedings*, volume 13, pages 41–64. Actes des Journées MODE, 2003.

[115] A. Schrijver. *Theory of Linear and Integer Programming*. J. Wiley and Sons, Chichester, England, 1986.

[116] D. F. Shanno. Conditioning of quasi-Newton methods for function minimization. *Mathematics of Computation*, 24:647–657, 1970.

[117] D. F. Shanno and K. H. Phua. Matrix conditionning and nonlinear optimization. *Mathematical Programming*, 14:149–160, 1978.

[118] T. Steihaug. The conjugate gradient method and trust regions in large scale optimization. *SIAM Journal on Numerical Analysis*, 20(3):626–637, 1983.

[119] Ph. L. Toint. Towards an efficient sparsity exploiting Newton method for minimization. In I. S. Duff, editor, *Sparse Matrices and Their Uses*, pages 57–88, London, 1981. Academic Press.

[120] Ph. L. Toint. Global convergence of a class of trust region methods for nonconvex minimization in Hilbert space. *IMA Journal of Numerical Analysis*, 8(2):231–252, 1988.

[121] Ph. L. Toint. A non-monotone trust-region algorithm for nonlinear optimization subject to convex constraints. *Mathematical Programming*, 77(1):69–94, 1997.

[122] M. Ulbrich, S. Ulbrich, and L. N. Vicente. A globally convergent primal-dual interior point filter method for nonconvex nonlinear programming. *Mathematical Programming, Series B*, 100(2):379–410, 2004.

[123] S. Ulbrich. On the superlinear local convergence of a filter-SQP method. *Mathematical Programming, Series B*, 100(1):217–245, 2004.

[124] A. Wächter. *An Interior Point Algorithm for Large-Scale Nonlinear Optimization with Applications in Process Engineering*. PhD thesis, Department of Chemical Engineering, Carnegie Mellon University, Pittsburgh, USA, 2002.

[125] A. Wächter and L. T. Biegler. Line search Filter methods for nonlinear programming: local convergence. *SIAM Journal on Optimization*, 16(1):32–48, 2005.

[126] A. Wächter and L. T. Biegler. Line search Filter methods for nonlinear programming: motivation and global convergence. *SIAM Journal on Optimization*, 16(1):1–31, 2005.

[127] R. B. Wilson. *A simplicial algorithm for concave programming*. PhD thesis, Harvard University, Massachusetts, USA, 1963.

[128] M. H. Wright. Direct search methods: once scorned, now respectable. In D. F. Griffiths and G. A. Watson, editors, *Proceedings of the 1995 Dundee Biennal Conference in Numerical Analysis*, Reading, Massachusetts, USA, 1996. Addison-Wesley Publishing Company.

[129] S. J. Wright. *Primal-Dual Interior-Point Methods*. SIAM, Philadelphia, USA, 1997.

[130] Y. Yuan, editor. *Advances in Nonlinear Programming*, Dordrecht, The Netherlands, 1998. Kluwer Academic Publishers.

[131] C. Zhu, R. H. Byrd, P. Lu, and J. Nocedal. Algorithm 778: L-BFGS-B: Fortran subroutines for large-scale bound constrained optimization. *ACM Transactions on Mathematical Software*, 23(4):550–560, 1997.

[132] C.A. Zoppke-Donaldson. *A tolerance-tube approach to sequential quadratic programming with applications*. PhD thesis, Department of Mathematics, University of Dundee, Dundee, Scotland, 1995.

# Main notations and abbreviations

General

| | |
|---|---|
| $\mathbb{N}$ | set of nonnegative integers |
| $\mathbb{R}$ | set of real numbers |
| $\mathbb{R}^n$ | real $n$-dimensional Euclidean space |
| $\|\cdot\|$ | absolute value of a scalar |
| $\|\|\cdot\|\|$ | vector norm (Euclidean unless otherwise specified) |
| $\|\mathcal{S}\|$ | cardinality of the set $\mathcal{S}$ |
| $\nabla_x f(x)$ | gradient of $f$ |
| $\nabla^2_{xx} f(x)$ | Hessian matrix of $f$ |
| $J_x c(x)$ | Jacobian of $c$ |
| $P[\cdot]$ | projection operator |
| $\varepsilon_M$ | machine precision |

Mathematical programming

| | |
|---|---|
| $\mathcal{E}$ | set of equality constraints |
| $\mathcal{I}$ | set of inequality constraints |
| $x^*$ | optimal solution |
| $\Omega$ | feasible region |
| strcit$\{\Omega\}$ | strictly feasible region |
| $\mathcal{N}$ | neighbourhood |
| $co\{\mathcal{S}\}$ | convex hull |
| Null($A$) | null space of $A$ |
| $\mathcal{A}(x)$ | active set at $x$ |
| $\mathcal{L}(x, \lambda)$ | Lagrangian function |
| $\mathcal{B}(x^*)$ | binding set at the solution |
| $\mathcal{B}_s(x^*)$ | strictly binding set at the solution |

Algorithms

| | |
|---|---|
| $x_k$ | $k$th iterate (vector) |
| $\mathcal{B}_k$ | trust region at iteration $k$ |
| $\Delta_k$ | trust-region radius at iteration $k$ |
| $g_k$ | gradient of the objective function at $x_k$ |
| $\bar{g}_k$ | "projected" gradient of the objective function at $x_k$ |
| $H_k$ | symmetric approximation to the objective Hessian at $x_k$ |
| $s_k$ | step at iteration $k$ |
| $x_k^C$ | (generalized) Cauchy point |
| $m_k(\cdot)$ | model of $f$ at the $k$th iteration |
| $\rho_k$ | ratio of actual to predicted decrease |
| $\mathcal{F}$ | filter |
| $\mathcal{S}$ | set of indices of successful iterations |
| $\mathcal{A}$ | set of indices of filter iterations |
| $\mathcal{D}$ | set of indices of sufficient descent iterations |
| $\mathcal{N}$ | set of indices of nonconvex iterations |

---

Main mathematical notations

| | |
|---|---|
| BFGS | Broyden-Fletcher-Goldfarb-Shanno |
| BTR | Basic Trust Region |
| CG | Conjugate Gradient |
| DFO | Derivative-Free Optimization |
| DFP | Davidon-Fletcher-Powell |
| EQP | Equality-constrained Quadratic Programming |
| GCP | Generalized Cauchy Point |
| GLTR | Generalized Lanczos Trust Region |
| KKT | Karush-Kuhn-Tucker |
| LICQ | Linear Independence Constraint Qualfication |
| MFCQ | Mangasarian-Fromovitz Constraint Qualfication |
| NLP | Nonlinear Programming |
| SLP | Sequential Linear Programming |
| SQP | Sequential Quadratic Programming |
| SR1 | Symmetric-Rank-One |

Main abbreviations

# Index


$\mathcal{N}_+(x, \lambda)$, 23

acceptable for the filter, 45, 54, 138
accumulation point, 10
achieved reduction, 30, 39
active constraint, *see* constraint
active set, 20, 130
active-set SQP methods, 39–40
actual reduction, *see* achieved reduction
algorithms
    augmented Lagrangian algorithm, 37
    basic trust-region algorithm (BTR), 31
    filter-trust-region algorithm, 56, 139
augmented Lagrangian
    function, 36
    method, 36–37
automatic differentiation, 15

backtracking, 29
binding set, 131
    strictly, 131
bound
    hard, 130
    simple, 130
    soft, 130
bound-constrained optimization, 4, 129–134
    problem, 130
bounded set, 8
breakpoint, 137

Cauchy
    arc, 31, 136
    point, 31, 32
Cauchy-Schwartz inequality, 7
chain rule, 12
closed set, 8
combined performance, 121
compact set, 8
complementary slackness condition, 22
concave, 9
conjugate
    directions, 32
    gradients, *see* conjugate-gradient methods
conjugate-gradient methods, 32–33, 137, 150
constrained optimization
    methods for, 33–40
    optimality conditions for, 20–24
    problems, **3**
constraint, 1
    active, 21, 130
    equality, **3**
    inactive, 21
    inequality, **3**, 43
    qualification, **21**
        linear independence (LICQ), 21
        Mangasarian-Fromovitz (MFCQ), 22
        Slater, 21
convergence, 10
    global, 11
    linear, 10
    local, 11
    order $r$, 10
    quadratic, 10
    superlinear, 10
convex
    function, 9, 22, 23

hull, 9
programming, **5**, 20
set, 9
critical point
first-order, **19**, 22, 60, 140
second-order, **19**, 70
strong second-order, 23
criticality measure, 24, 134
curvature, **12**
direction of negative curvature, 12
direction of positive curvature, 12

derivative
first, 11
first partial, 11
second partial, 12
derivative-free optimization, 111
descent direction, 25
differentiability, 11
continuously differentiable function, 11
differentiable function, 11
twice-continuously differentiable function, 12
discrete optimization, 4
dominance, 42, 43, 54, 138
dual variables, 21

feasibility condition, 22
feasible
box, 130
point, **3**, 23, 130
region, **3**, 22
convex, 22
filter, 41, 42, **43**, 54–56, 138–139
algorithms, 56, 139
methods, 41–48
for bound-constrained optimization, 129–145
for unconstrained optimization, 51–72
multidimensional, 53, 54, 135, 138
finite differences, 13–15, 105–111
central, 14
forward, 14
first-order conditions, 19, 22, 131
free variable, 131

generalized Cauchy point, 136, 137, 157
globalization techniques, 24, **28**, 29, 39, 133
GLTR subroutine, 77
gradient, **11**
gradient-projection method, 132–134, 150

Hessian matrix, **12**
approximate, 26

indefinite matrix, 7
infeasible problem, 3
inner product, 6
integer programming, 4
interior-point methods, 5

Jacobian, **12**

Karush-Kuhn-Tucker (KKT)
conditions, **22**, 23
point, 22

Lagrange
function, *see* Lagrangian
multipliers, **21**, 22, 23
Lagrangian, **21**
augmented, *see* augmented Lagrangian
limit
inferior, 10
point, 10
superior, 10
line search
exact, 29
inexact, 29
line-search methods, 29
linear

program, 4
programming, 4
linear conjugate-gradient methods, 32
linear independence
constraint qualification (LICQ), 21
Lipschitz continuous, 9
locally, 9

margin, 45, 54
mathematical
program, **2**, 3–6
programming, 2–6
mean value theorem, 13
merit function, 34, 38, 39, 41, 43
minimizer
global, **17**, 20
local, **18**, 19, 20, 51, 131
strict local, 18
minimum, 18
model, 2
linear, 13
quadratic, 13, 25, 26, 30, 102, 136, 137
modelling, 2

neighbourhood, 8
Newton
direction, 25
equations, 25
step, 25
Newton's method
for unconstrained optimization, 25–26, 52
nondifferentiable optimization, 5
nonlinear programming, **5**, 24–40
norm
Euclidean, 7
Frobenius, 8
matrix, 7
vector, 7
null space, 8

objective function, **1**, 3
open set, 8
operations research, 2
optimality
conditions, 18
for bound-constrained optimization, 130, 131
for constrained optimization, 20–24
for unconstrained optimization, 19–20
optimization, **1**, 1–15
constrained, *see* constrained optimization
unconstrained, *see* unconstrained optimization
orthogonal, 6

penalty
$\ell_1$ exact, 35
function, 34, 41, 44, 45
methods, 34–36
parameter, 34, 36, 44
quadratic, 35
performance profiles, 76–77
piecewise linear path, 132
polytope, 4
positive definite matrix, **6**, 25
positive semidefinite matrix, 6
pre-filtering, 93
predicted reduction, 31
primal variables, 21
programming, 2
projected gradient, 134
projected-gradient path, 132
projection operator, 132

quadratic
model, *see* model
quadratic programming, 5, 38
Quasi-Newton equation, 27
Quasi-Newton methods

for unconstrained optimization, 26–28, 113–119

restoration phase, 46
restricted
gradient, 131
Hessian, 131

saddle point, 20
secant
equation, 27
methods, *see* quasi-Newton methods
secant approximations, 113–119
second-order conditions, 19, 20, 23, 131
sequential quadratic programming (SQP), **38**, 38–40, 44
signed filter entries, 94, 151, 162
simplex method, 4
Slater's constraint qualification, 21
softwares
FILTBOUND, 147, 152
FILTRANE, 52
FILTRUNC, 58, 73, 80, 191
IPOPT, 134
KNITRO, 134
LANCELOT, 37, 73, 87, 133, 147, 157
LBFGS-B, 134
LOQO, 134
SNOPT, 39, 40
TRON, 134
filterSQP, 40, 46
solution
global, 22, 23
local, 22, 23, 131
standard form, 4
stationarity condition, 22
stochastic programming, 5
symmetric matrix, 6

Taylor approximation
first-order, 13
sesond-order, 13
transpose, 6
trust region, **30**, 52, 135
trust-region
methods, 29
for constrained problems, 39
for unconstrained problems, 29–32
radius, 30
subproblem, 30

unconstrained optimization
methods for, 24–33, 51–72
optimality conditions for, 19–20
problems, **3**
unsigned filter entries, 79, 94, 151, 162
updating formula, **26**
BFGS, 27, 113–115
DFP, 27
limited memory BFGS, 28
SR1, 27, 115–119

variable, 1
vector, 6

working set, 39, 132

# Appendix A

# Results of FILTRUNC

Tables A.1-A.6 document the runs for the *filter* and *pure trust-region* variants of our code FILTRUNC on the unconstrained problems from the CUTEr collection. For each test problem, the tables list the following characteristics :

- number of variables ($n$);
- number of iterations (# iter);
- number of conjugate-gradient iterations (# cgiter);
- required CPU time in seconds (CPU);
- final value of the objective function ($f(x^*)$).

For the filter variant, we also indicate the maximum number of filter entries (# nfilt). Note that the symbol - indicates that the variant failed to solve the problem within the prescribed iteration and CPU limitations.

| | | *filter* variant | | | | | *pure trust-region* variant | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Problem | $n$ | # iter | # cgiter | # nfilt | CPU | $f(x^*)$ | # iter | # cgiter | CPU | $f(x^*)$ |
| AIRCRFTB | 5 | 21 | 78 | 1 | 0.0020 | 6.8926E-14 | 18 | 56 | 0.0014 | 5.8503E-15 |
| ALLINITU | 4 | 7 | 21 | 0 | 0.0007 | 5.7444E+00 | 7 | 21 | 0.0007 | 5.7444E+00 |
| ARGLINA | 200 | 1 | 1 | 1 | 1.9893 | 2.0000E+02 | 5 | 5 | 2.0182 | 2.0000E+02 |
| ARGLINB | 200 | - | - | - | - | - | - | - | - | - |
| ARGLINC | 200 | - | - | - | - | - | - | - | - | - |
| ARWHEAD | 5000 | 5 | 6 | 0 | 0.1044 | 1.1100E-12 | 5 | 6 | 0.1043 | 1.1100E-12 |
| BARD | 3 | 13 | 34 | 2 | 0.0036 | 8.2149E-03 | 11 | 25 | 0.0032 | 8.2149E-03 |
| BDQRTIC | 5000 | 12 | 80 | 2 | 0.6161 | 2.0006E+04 | 13 | 84 | 0.6421 | 2.0006E+04 |
| BEALE | 2 | 9 | 14 | 0 | 0.0006 | 6.2978E-15 | 8 | 14 | 0.0006 | 6.2978E-15 |
| BIGGS3 | 3 | 14 | 37 | 1 | 0.0014 | 4.1398E-17 | 9 | 27 | 0.0010 | 4.1100E-17 |
| BIGGS5 | 5 | 30 | 136 | 2 | 0.0043 | 3.7511E-11 | 44 | 169 | 0.0054 | 8.7526E-13 |
| BIGGS6 | 6 | 410 | 775 | 4 | 0.0380 | 2.4268E-01 | 273 | 737 | 0.0277 | 2.4269E-01 |
| BOX2 | 2 | 6 | 7 | 0 | 0.0005 | 1.6056E-13 | 6 | 7 | 0.0005 | 1.6056E-13 |
| BOX3 | 3 | 7 | 19 | 1 | 0.0006 | 6.5695E-12 | 7 | 18 | 0.0006 | 2.3700E-12 |
| BRKMCC | 2 | 3 | 6 | 0 | 0.0003 | 1.6904E-01 | 3 | 6 | 0.0003 | 1.6904E-01 |
| BROWNAL | 200 | 2 | 3 | 1 | 0.9897 | 1.4731E-09 | 5 | 6 | 1.0195 | 1.4731E-09 |
| BROWNBS | 2 | 4 | 5 | 1 | 0.0004 | 1.9722E-31 | 25 | 39 | 0.0022 | 0.0000E+00 |
| BROWNDEN | 4 | 9 | 28 | 6 | 0.0008 | 8.5822E+04 | 11 | 35 | 0.0010 | 8.5822E+04 |
| BRYBND | 5000 | 11 | 89 | 0 | 0.5224 | 8.6725E-13 | 11 | 70 | 0.3715 | 9.3465E-13 |
| CHAINWOO | 4000 | - | - | - | - | - | - | - | - | - |
| CHNROSNB | 50 | 41 | 558 | 30 | 0.0133 | 1.2602E-13 | 60 | 723 | 0.0213 | 4.5672E-16 |
| CLIFF | 2 | 27 | 28 | 1 | 0.0010 | 1.9979E-01 | 28 | 31 | 0.0011 | 1.9979E-01 |
| CLPLATEA | 10100 | 20 | 1285 | 7 | 10.9350 | -1.2931E-02 | 15 | 1130 | 6.3750 | -1.2931E-02 |
| CLPLATEB | 4970 | 2 | 420 | 1 | 1.6702 | -5.0512E-03 | 4 | 491 | 2.4030 | -5.0512E-03 |
| CLPLATEC | 4970 | 2 | 7680 | 1 | 30.9090 | -5.0147E-03 | 4 | 13072 | 71.6170 | -5.0147E-03 |
| COSINE | 10000 | 11 | 21 | 0 | 0.2435 | -9.9990E+03 | 11 | 16 | 0.2007 | -9.9990E+03 |
| CRAGGLVY | 5000 | 14 | 146 | 8 | 0.4487 | 1.6882E+03 | 16 | 138 | 0.4443 | 1.6882E+03 |

Table A.1: Numerical results for the *filter* and *pure trust-region* variants of FILTRUNC (continued on next page)

| | | *filter* variant | | | | | *pure trust-region* variant | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Problem | $n$ | # iter | # cgiter | # nfilt | CPU | $f(x^*)$ | # iter | # cgiter | CPU | $f(x^*)$ |
| CUBE | 2 | 31 | 51 | 8 | 0.0013 | 7.3587E-20 | 36 | 60 | 0.0015 | 2.5721E-20 |
| CURLY10 | 10000 | 14 | 62714 | 4 | 202.6020 | -1.0032E+06 | 16 | 62674 | 250.4370 | -1.0032E+06 |
| CURLY20 | 10000 | 15 | 75234 | 2 | 312.0390 | -1.0032E+06 | 17 | 75432 | 334.5440 | -1.0032E+06 |
| CURLY30 | 1000 | 16 | 7397 | 1 | 3.4927 | -1.0032E+05 | 17 | 7007 | 3.4300 | -1.0032E+05 |
| DECONVU | 61 | 22 | 360 | 1 | 0.0514 | 2.9591E-08 | 23 | 407 | 0.0516 | 1.7111E-08 |
| DENSCHNA | 2 | 6 | 12 | 0 | 0.0004 | 1.1028E-23 | 6 | 12 | 0.0004 | 1.1028E-23 |
| DENSCHNB | 2 | 3 | 6 | 0 | 0.0004 | 1.5777E-30 | 3 | 5 | 0.0004 | 1.5777E-30 |
| DENSCHNC | 2 | 11 | 21 | 1 | 0.0006 | 1.1329E-22 | 11 | 19 | 0.0007 | 9.9024E-15 |
| DENSCHND | 3 | 34 | 95 | 0 | 0.0018 | 2.5550E-09 | 34 | 93 | 0.0017 | 2.5644E-09 |
| DENSCHNE | 3 | 19 | 23 | 1 | 0.0010 | 1.8548E-14 | 18 | 23 | 0.0010 | 1.6624E-14 |
| DENSCHNF | 2 | 6 | 12 | 0 | 0.0005 | 6.5132E-22 | 6 | 12 | 0.0005 | 6.5132E-22 |
| DIXMAANA | 9000 | 11 | 25 | 0 | 0.3009 | 1.0000E+00 | 11 | 20 | 0.2451 | 1.0000E+00 |
| DIXMAANB | 9000 | 7 | 12 | 1 | 0.1614 | 1.0000E+00 | 11 | 19 | 0.2264 | 1.0000E+00 |
| DIXMAANC | 9000 | 8 | 16 | 1 | 0.1948 | 1.0000E+00 | 12 | 20 | 0.2668 | 1.0000E+00 |
| DIXMAAND | 9000 | 9 | 19 | 1 | 0.2210 | 1.0000E+00 | 13 | 23 | 0.2698 | 1.0000E+00 |
| DIXMAANE | 9000 | 13 | 391 | 0 | 1.8045 | 1.0000E+00 | 13 | 373 | 1.6234 | 1.0000E+00 |
| DIXMAANF | 9000 | 17 | 442 | 2 | 3.4257 | 1.0000E+00 | 27 | 451 | 3.4543 | 1.0000E+00 |
| DIXMAANG | 9000 | 22 | 438 | 1 | 3.1785 | 1.0000E+00 | 26 | 481 | 3.8567 | 1.0000E+00 |
| DIXMAANH | 9000 | 19 | 380 | 2 | 2.5920 | 1.0000E+00 | 28 | 449 | 3.5217 | 1.0000E+00 |
| DIXMAANI | 9000 | 14 | 7788 | 0 | 28.2500 | 1.0000E+00 | 14 | 7667 | 27.7570 | 1.0000E+00 |
| DIXMAANJ | 9000 | 31 | 718 | 2 | 9.2960 | 1.0000E+00 | 40 | 640 | 7.0225 | 1.0000E+00 |
| DIXMAANK | 9000 | 35 | 893 | 2 | 11.7360 | 1.0000E+00 | 42 | 913 | 10.3870 | 1.0000E+00 |
| DIXMAANL | 9000 | 31 | 624 | 3 | 8.0215 | 1.0000E+00 | 38 | 605 | 6.5845 | 1.0000E+00 |
| DIXON3DQ | 10000 | 3 | 17065 | 3 | 34.3660 | 2.1775E-12 | 9 | 22563 | 69.9030 | 2.0487E-09 |
| DJTL | 2 | 106 | 158 | 5 | 0.0054 | -8.9515E+03 | 111 | 164 | 0.0054 | -8.9515E+03 |
| DQDRTIC | 5000 | 4 | 11 | 1 | 0.0382 | 6.7298E-12 | 11 | 18 | 0.0787 | 1.4185E-13 |
| DQRTIC | 5000 | 33 | 202 | 28 | 0.4184 | 3.6250E-06 | 48 | 196 | 0.3001 | 3.5818E-06 |

Table A.2: Numerical results for the *filter* and *pure trust-region* variants of FILTRUNC (continued on next page)

| | | *filter* variant | | | | | *pure trust-region* variant | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Problem | $n$ | # iter | # cgiter | # nfilt | CPU | $f(x^*)$ | # iter | # cgiter | CPU | $f(x^*)$ |
| EDENSCH | 10000 | 12 | 29 | 5 | 0.3480 | 6.0003E+04 | 19 | 47 | 0.5208 | 6.0003E+04 |
| EG2 | 1000 | 3 | 3 | 0 | 0.0052 | -9.9895E+02 | 3 | 3 | 0.0052 | -9.9895E+02 |
| EIGENALS | 2550 | 44 | 1838 | 8 | 112.6 | 4.2691E-10 | 54 | 1900 | 119.5 | 6.0437E-10 |
| EIGENBLS | 2550 | 262 | 33950 | 85 | 2139.3 | 4.9966E-09 | 522 | 32065 | 2126.6 | 5.3324E-06 |
| EIGENCLS | 2652 | 518 | 28155 | 26 | 2097.1 | 2.7628E-11 | 808 | 48811 | 3549.6 | 1.2440E-10 |
| ENGVAL1 | 10000 | 8 | 25 | 2 | 0.2448 | 1.1099E+04 | 13 | 30 | 0.3153 | 1.1099E+04 |
| ENGVAL2 | 2 | 13 | 38 | 0 | 0.0008 | 2.4805E-16 | 13 | 38 | 0.0008 | 2.4805E-16 |
| ERRINROS | 50 | 55 | 549 | 16 | 0.0187 | 3.9904E+01 | 66 | 653 | 0.0238 | 3.9904E+01 |
| EXPFIT | 2 | 9 | 16 | 0 | 0.0007 | 2.4051E-01 | 9 | 16 | 0.0007 | 2.4051E-01 |
| EXTROSNB | 1000 | 56 | 407 | 26 | 0.1585 | 1.9242E-06 | 267 | 2300 | 6.4000 | 6.8479E-06 |
| FMINSRF2 | 5625 | - | - | - | - | - | 639 | 4752 | 32.3660 | 1.0000E+00 |
| FMINSURF | 49 | - | - | - | - | - | 15 | 114 | 0.0057 | 5.4376E+02 |
| FREUROTH | 5000 | 16 | 41 | 1 | 0.2700 | 6.0816E+05 | 67 | 73 | 0.7300 | 6.0816E+05 |
| GENROSE | 500 | 533 | 5328 | 3 | 1.5157 | 1.0000E+00 | 360 | 4068 | 1.0911 | 1.0000E+00 |
| GROWTHLS | 3 | 160 | 304 | 4 | 0.0125 | 1.0040E+00 | 139 | 283 | 0.0105 | 1.0040E+00 |
| GULF | 3 | 29 | 59 | 2 | 0.0131 | 1.3258E-12 | 32 | 65 | 0.0174 | 1.2193E-10 |
| HAIRY | 2 | 104 | 184 | 2 | 0.0051 | 2.0000E+01 | 105 | 184 | 0.0051 | 2.0000E+01 |
| HATFLDD | 3 | 20 | 57 | 1 | 0.0015 | 6.6151E-08 | 22 | 55 | 0.0015 | 6.6151E-08 |
| HATFLDE | 3 | 20 | 55 | 0 | 0.0019 | 5.1204E-07 | 20 | 55 | 0.0019 | 5.1204E-07 |
| HEART6LS | 6 | 921 | 3521 | 6 | 0.1045 | 7.1941E-14 | 968 | 3265 | 0.0871 | 5.6037E-12 |
| HEART8LS | 8 | 98 | 579 | 0 | 0.0155 | 5.0318E-17 | 98 | 575 | 0.0146 | 1.9061E-18 |
| HELIX | 3 | 11 | 27 | 0 | 0.0008 | 6.2667E-18 | 11 | 27 | 0.0008 | 6.2667E-18 |
| HIELOW | 3 | 9 | 23 | 0 | 0.1630 | 8.7417E+02 | 9 | 22 | 0.1621 | 8.7417E+02 |
| HILBERTA | 2 | 1 | 2 | 1 | 0.0003 | 2.6046E-28 | 3 | 5 | 0.0004 | 7.3956E-32 |
| HILBERTB | 10 | 3 | 6 | 1 | 0.0005 | 6.7627E-17 | 6 | 11 | 0.0007 | 8.5991E-18 |
| HIMMELBB | 2 | 7 | 11 | 0 | 0.0006 | 2.9031E-17 | 7 | 11 | 0.0006 | 2.9031E-17 |
| HIMMELBF | 4 | 107 | 424 | 0 | 0.0087 | 3.1857E+02 | 107 | 422 | 0.0086 | 3.1857E+02 |

Table A.3: Numerical results for the *filter* and *pure trust-region* variants of FILTRUNC (continued on next page)

| | | *filter* variant | | | | | *pure trust-region* variant | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Problem | $n$ | # iter | # cgiter | # nfilt | CPU | $f(x^*)$ | # iter | # cgiter | CPU | $f(x^*)$ |
| HIMMELBG | 2 | 6 | 11 | 0 | 0.0005 | 1.3959E-22 | 6 | 11 | 0.0005 | 1.3959E-22 |
| HIMMELBH | 2 | 4 | 6 | 0 | 0.0004 | -1.0000E+00 | 4 | 6 | 0.0004 | -1.0000E+00 |
| HYDC20LS | 99 | - | - | - | - | - | - | - | - | - |
| JENSMP | 2 | 10 | 20 | 1 | 0.0007 | 1.2436E+02 | 16 | 21 | 0.0009 | 1.2436E+02 |
| KOWOSB | 4 | 11 | 35 | 2 | 0.0011 | 3.0780E-04 | 11 | 35 | 0.0011 | 3.0780E-04 |
| LIARWHD | 5000 | 13 | 24 | 7 | 0.1449 | 9.0726E-21 | 16 | 29 | 0.1615 | 1.0691E-22 |
| LMINSURF | 5329 | - | - | - | - | - | - | - | - | - |
| LOGHAIRY | 2 | - | - | - | - | - | - | - | - | - |
| MANCINO | 100 | 17 | 69 | 0 | 0.8219 | 3.4970E-20 | 18 | 41 | 0.8962 | 2.2697E-21 |
| MARATOSB | 2 | 743 | 1101 | 3 | 0.0309 | -1.0000E+00 | - | - | - | - |
| MEXHAT | 2 | 22 | 32 | 4 | 0.0009 | -4.0010E-02 | 28 | 38 | 0.0012 | -4.0010E-02 |
| MEYER3 | 3 | - | - | - | - | - | - | - | - | - |
| MINSURF | 36 | - | - | - | - | - | 8 | 36 | 0.0025 | 1.0000E+00 |
| MOREBV | 5000 | 1 | 98 | 0 | 0.1464 | 2.1341E-10 | 1 | 98 | 0.1474 | 2.1341E-10 |
| MSQRTALS | 1024 | 42 | 8394 | 0 | 41.1135 | 7.9382E+03 | 40 | 6211 | 24.6770 | 7.9382E+03 |
| MSQRTBLS | 1024 | 33 | 6135 | 0 | 27.2500 | 7.9264E+03 | 32 | 4276 | 20.2610 | 7.9264E+03 |
| NCB20 | 5010 | 75 | 740 | 3 | 30.1680 | -1.4661E+03 | 58 | 564 | 24.3380 | -1.4566E+03 |
| NCB20B | 5000 | 30 | 1334 | 1 | 57.0450 | 7.3513E+03 | 29 | 916 | 36.4690 | 7.3513E+03 |
| NLMSURF | 5329 | - | - | - | - | - | - | - | - | - |
| NONCVXU2 | 5000 | - | - | - | - | - | - | - | - | - |
| NONCVXUN | 5000 | - | - | - | - | - | - | - | - | - |
| NONDIA | 5000 | 4 | 5 | 1 | 0.0408 | 9.3264E-09 | 5 | 6 | 0.0467 | 9.2570E-09 |
| NONDQUAR | 5000 | 53 | 1502 | 14 | 1.8023 | 2.5472E-06 | 38 | 439 | 1.0087 | 4.6370E-05 |
| NONMSQRT | 100 | 77 | 2464 | 24 | 0.3817 | 1.8054E+01 | 320 | 1132 | 2.3160 | 1.8054E+01 |
| ODC | 4900 | 29 | 2218 | 1 | 7.2580 | -1.1372E-02 | 30 | 2261 | 7.4595 | -1.1372E-02 |
| OSBORNEA | 5 | 22 | 81 | 6 | 0.0030 | 5.4649E-05 | 50 | 181 | 0.0072 | 5.4649E-05 |
| OSBORNEB | 11 | 16 | 142 | 0 | 0.0128 | 4.0138E-02 | 16 | 140 | 0.0125 | 4.0138E-02 |

Table A.4: Numerical results for the *filter* and *pure trust-region* variants of FILTRUNC (continued on next page)

| | | *filter* variant | | | | | *pure trust-region* variant | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Problem | $n$ | # iter | # cgiter | # nfilt | CPU | $f(x^*)$ | # iter | # cgiter | CPU | $f(x^*)$ |
| PALMER1C | 8 | 5 | 28 | 4 | 0.0007 | 9.7605E-02 | 11 | 71 | 0.0017 | 9.7605E-02 |
| PALMER1D | 7 | 4 | 19 | 3 | 0.0006 | 6.5267E-01 | 17 | 58 | 0.0017 | 6.5267E-01 |
| PALMER2C | 8 | 4 | 24 | 4 | 0.0006 | 1.4369E-02 | 9 | 61 | 0.0013 | 1.4369E-02 |
| PALMER3C | 8 | 4 | 24 | 4 | 0.0006 | 1.9538E-02 | 9 | 61 | 0.0014 | 1.9538E-02 |
| PALMER4C | 8 | 4 | 23 | 4 | 0.0006 | 5.0311E-02 | 11 | 72 | 0.0016 | 5.0311E-02 |
| PALMER5C | 6 | 4 | 14 | 1 | 0.0005 | 2.1281E+00 | 9 | 24 | 0.0007 | 2.1281E+00 |
| PALMER6C | 8 | 4 | 23 | 3 | 0.0005 | 1.6387E-02 | 10 | 71 | 0.0014 | 1.6387E-02 |
| PALMER7C | 8 | 4 | 31 | 4 | 0.0006 | 6.0199E-01 | 13 | 98 | 0.0018 | 6.0199E-01 |
| PALMER8C | 8 | 4 | 27 | 4 | 0.0005 | 1.5977E-01 | 10 | 71 | 0.0014 | 1.5977E-01 |
| PARKCH | 15 | 28 | 257 | 4 | 126.1790 | 1.6237E+03 | 24 | 216 | 101.1600 | 1.6237E+03 |
| PENALTY1 | 1000 | 37 | 44 | 3 | 0.6537 | 9.6862E-03 | 56 | 69 | 0.6752 | 9.6862E-03 |
| PENALTY2 | 200 | 12 | 283 | 2 | 0.0346 | 4.7116E+13 | 54 | 332 | 0.0683 | 4.7116E+13 |
| PENALTY3 | 200 | 25 | 110 | 1 | 3.2267 | 1.0010E-03 | 35 | 101 | 3.2765 | 9.9928E-04 |
| POWELLSG | 5000 | 16 | 63 | 9 | 0.1273 | 1.2357E-06 | 20 | 71 | 0.1322 | 7.0643E-07 |
| POWER | 100 | 23 | 203 | 1 | 0.0071 | 1.9404E-09 | 24 | 216 | 0.0075 | 1.8990E-09 |
| QUARTC | 5000 | 33 | 202 | 28 | 0.4159 | 3.6250E-06 | 48 | 196 | 0.3019 | 3.5818E-06 |
| RAYBENDL | 2046 | 252 | 114390 | 0 | 186.3400 | 9.6242E+01 | - | - | - | - |
| RAYBENDS | 2046 | 315 | 27889 | 103 | 165.0300 | 9.6242E+01 | - | - | - | - |
| ROSENBR | 2 | 22 | 38 | 7 | 0.0011 | 9.1504E-15 | 27 | 48 | 0.0012 | 8.0428E-22 |
| S308 | 2 | 10 | 20 | 1 | 0.0006 | 7.7320E-01 | 10 | 18 | 0.0006 | 7.7320E-01 |
| SBRYBND | 500 | - | - | - | - | - | - | - | - | - |
| SCHMVETT | 5000 | 4 | 48 | 2 | 0.2589 | -1.4994E+04 | 7 | 45 | 0.3088 | -1.4994E+04 |
| SCOSINE | 5000 | - | - | - | - | - | - | - | - | - |
| SCURLY10 | 100 | - | - | - | - | - | - | - | - | - |
| SCURLY20 | 100 | 582 | 451200 | 217 | 19.0700 | -1.0032E+04 | 452 | 404859 | 14.3700 | -1.0032E+04 |
| SCURLY30 | 100 | 906 | 750334 | 221 | 34.7020 | -1.0032E+04 | 694 | 638656 | 25.8390 | -1.0032E+04 |
| SENSORS | 100 | 29 | 145 | 0 | 0.9086 | -1.9668E+03 | 29 | 139 | 0.8847 | -1.9668E+03 |

Table A.5: Numerical results for the *filter* and *pure trust-region* variants of FILTRUNC (continued on next page)

Table A.6: Numerical results for the *filter* and *pure trust-region* variants of FILTRUNG

| | | *filter* variant | | | | | *pure trust-region* variant | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Problem | $n$ | # iter | # cgiter | # nfilt | CPU | $f(x^*)$ | # iter | # cgiter | CPU | $f(x^*)$ |
| SINEVAL | 2 | 9 | 12 | 3 | 0.0005 | 1.3924E-16 | 59 | 100 | 0.0024 | 1.1805E-15 |
| SINQUAD | 10000 | 16 | 40 | 1 | 0.7977 | -2.6423E+07 | 16 | 35 | 0.7149 | -2.6423E+07 |
| SISSER | 2 | 14 | 25 | 0 | 0.0007 | 4.2061E-10 | 14 | 25 | 0.0007 | 4.2061E-10 |
| SNAIL | 2 | 77 | 113 | 2 | 0.0029 | 1.0215E-19 | 62 | 120 | 0.0026 | 1.9560E-17 |
| SPARSINE | 5000 | 35 | 81833 | 3 | 272.4800 | 2.2832E-10 | 49 | 100999 | 458.3750 | 3.1017E-11 |
| SPARSQUR | 10000 | 20 | 135 | 4 | 1.7432 | 1.2227E-07 | 23 | 120 | 1.7105 | 1.3168E-07 |
| SPMSRTLS | 4900 | 25 | 547 | 2 | 3.2902 | 3.4634E-11 | 19 | 416 | 1.7560 | 5.8338E-10 |
| SROSENBR | 5000 | 7 | 12 | 2 | 0.0470 | 7.6325E-12 | 9 | 15 | 0.0553 | 1.0349E-15 |
| SSC | 4900 | 2 | 141 | 1 | 0.9595 | -2.0782E+00 | 4 | 158 | 1.2306 | -2.0782E+00 |
| STRATEC | 10 | 28 | 198 | 5 | 52.1180 | 2.2123E+03 | 31 | 205 | 56.1620 | 2.2123E+03 |
| TESTQUAD | 5000 | 6 | 1769 | 4 | 1.8488 | 4.2524E-13 | 11 | 1787 | 1.8563 | 4.2353E-13 |
| TOINTGOR | 50 | 7 | 152 | 3 | 0.0026 | 1.3739E+03 | 9 | 157 | 0.0033 | 1.3739E+03 |
| TOINTGSS | 5000 | 4 | 12 | 1 | 0.0862 | 1.0004E+01 | 13 | 26 | 0.1850 | 1.0002E+01 |
| TOINTPSP | 50 | 85 | 316 | 40 | 0.0132 | 2.2556E+02 | 17 | 103 | 0.0030 | 2.2556E+02 |
| TOINTQOR | 50 | 4 | 44 | 1 | 0.0010 | 1.1755E+03 | 7 | 56 | 0.0016 | 1.1755E+03 |
| TQUARTIC | 5000 | 1 | 2 | 1 | 0.0156 | 5.5847E-21 | 11 | 18 | 0.0938 | 1.8857E-16 |
| TRIDIA | 5000 | 5 | 1124 | 2 | 1.0509 | 7.2945E-14 | 11 | 1315 | 1.2708 | 1.2573E-13 |
| VARDIM | 200 | 29 | 29 | 3 | 0.0151 | 1.1665E-24 | 29 | 29 | 0.0151 | 4.7159E-19 |
| VAREIGVL | 50 | 16 | 501 | 2 | 0.0193 | 7.1497E-11 | 16 | 290 | 0.0150 | 1.6682E-10 |
| VIBRBEAM | 8 | 46 | 196 | 7 | 0.0155 | 1.7489E+00 | 59 | 265 | 0.0240 | 1.7489E+00 |
| WATSON | 12 | 9 | 48 | 0 | 0.0033 | 8.0778E-10 | 9 | 49 | 0.0035 | 8.0312E-10 |
| WOODS | 10000 | 53 | 169 | 11 | 1.2684 | 1.2982E-13 | 58 | 180 | 1.2191 | 3.4171E-12 |
| YFITU | 3 | 42 | 104 | 17 | 0.0033 | 6.4285E-10 | 66 | 157 | 0.0051 | 6.6720E-13 |
| ZANGWIL2 | 2 | 1 | 1 | 1 | 0.0003 | -1.8200E+01 | 2 | 2 | 0.0003 | -1.8200E+01 |

# Appendix B

# Results of FILTBOUND

Tables B.1-B.4 document the runs for the *filter* and *pure trust-region* variants of our code FILTBOUND on the bound-constrained problems from the CUTEr collection. For each test problem, the tables list the following characteristics :

- number of variables ($n$);
- number of iterations (# iter);
- number of conjugate-gradient iterations (# cgiter);
- required CPU time in seconds (CPU);
- final value of the objective function ($f(x^*)$).

For the filter variant, we also indicate the maximum number of filter entries (# nfilt). Note that the symbol - indicates that the variant failed to solve the problem within the prescribed iteration and CPU time limitations.

| | | *filter* variant | | | | | *pure trust-region* variant | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Problem | $n$ | # iter | # cgiter | # nfilt | CPU | $f(x^*)$ | # iter | # cgiter | CPU | $f(x^*)$ |
| 3PK | 30 | 12 | 459 | 7 | 0.0108 | 1.7201E+00 | 18 | 544 | 0.0127 | 1.7201E+00 |
| ALLINIT | 3 | 7 | 13 | 1 | 0.0009 | 1.6706E+01 | 6 | 15 | 0.0008 | 1.6706E+01 |
| BDEXP | 5000 | 13 | 73 | 0 | 0.4006 | 2.4047E-04 | 12 | 69 | 0.3868 | 2.4047E-04 |
| BIGGSB1 | 5000 | - | - | - | - | - | - | - | - | - |
| BQP1VAR | 1 | 1 | 0 | 0 | 0.0003 | 0.0000E+00 | 1 | 0 | 0.0003 | 0.0000E+00 |
| BQPGABIM | 46 | 3 | 25 | 0 | 0.0021 | -3.7903E-05 | 3 | 24 | 0.0021 | -3.7903E-05 |
| BQPGASIM | 50 | 3 | 22 | 0 | 0.0020 | -5.5198E-05 | 3 | 22 | 0.0020 | -5.5198E-05 |
| BQPGAUSS | 2003 | 6 | 8966 | 0 | 13.7020 | -3.6258E-01 | 6 | 8966 | 13.7700 | -3.6258E-01 |
| CAMEL6 | 2 | 6 | 8 | 0 | 0.0006 | -1.0316E+00 | 6 | 7 | 0.0006 | -1.0316E+00 |
| CHEBYQAD | 100 | 75 | 3117 | 19 | 3.0350 | 8.7162E-03 | 115 | 5859 | 5.3150 | 8.7158E-03 |
| CHENHARK | 5000 | 304 | 1103584 | 0 | 2109.0649 | -2.0000E+00 | 301 | 1107770 | 2111.1121 | -2.0000E+00 |
| CVXBQP1 | 10000 | 1 | 1 | 0 | 19.6260 | 2.2502E+06 | 1 | 1 | 19.4990 | 2.2502E+06 |
| DECONVB | 61 | 18 | 637 | 1 | 0.0521 | 4.2827E-09 | 14 | 250 | 0.0251 | 5.5886E-12 |
| EG1 | 3 | 5 | 5 | 0 | 0.0006 | -1.1328E+00 | 5 | 5 | 0.0006 | -1.1328E+00 |
| EXPLIN | 1200 | 15 | 405 | 1 | 1.2072 | -7.1925E+07 | 31 | 325 | 1.3283 | -7.1925E+07 |
| EXPLIN2 | 1200 | 12 | 199 | 1 | 1.1440 | -7.1999E+07 | 12 | 191 | 1.1184 | -7.1999E+07 |
| EXPQUAD | 1200 | 160 | 884 | 31 | 13.0540 | -3.6849E+09 | 134 | 208 | 33.7250 | -3.6849E+09 |
| HADAMALS | 380 | 9 | 34 | 1 | 0.1022 | 7.3118E+03 | 9 | 24 | 0.0985 | 7.3118E+03 |
| HART6 | 6 | 9 | 18 | 0 | 0.0013 | -3.3229E+00 | 9 | 18 | 0.0013 | -3.3229E+00 |
| HATFLDA | 4 | 29 | 66 | 1 | 0.0025 | 2.4401E-12 | 28 | 71 | 0.0026 | 3.3473E-14 |
| HATFLDB | 4 | 25 | 44 | 1 | 0.0020 | 5.5728E-03 | 25 | 52 | 0.0021 | 5.5728E-03 |
| HATFLDC | 25 | 4 | 33 | 0 | 0.0013 | 7.4939E-14 | 4 | 33 | 0.0013 | 7.4939E-14 |
| HIMMELP1 | 2 | 10 | 5 | 0 | 0.0008 | -6.2054E+01 | 10 | 5 | 0.0008 | -6.2054E+01 |
| HS1 | 2 | 11 | 14 | 3 | 0.0010 | 5.5402E-15 | 33 | 45 | 0.0022 | 5.3596E-17 |
| HS110 | 200 | 7 | 0 | 1 | 0.0008 | -4.5778E+01 | 7 | 0 | 0.0008 | -4.5778E+01 |
| HS2 | 2 | 6 | 3 | 0 | 0.0006 | 4.9412E+00 | 6 | 3 | 0.0006 | 4.9412E+00 |
| HS25 | 3 | 0 | 0 | 0 | 0.0003 | 3.283E+01 | 0 | 0 | 0.0003 | 3.283E+01 |

Table B.1: Numerical results for the *filter* and *pure trust-region* variants of FILTBOUND (continued on next page)

| | | *filter* variant | | | | | *pure trust-region* variant | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Problem | $n$ | # iter | # cgiter | # nfilt | CPU | $f(x^*)$ | # iter | # cgiter | CPU | $f(x^*)$ |
| HS3 | 2 | 1 | 0 | 1 | 0.0003 | 2.1065E-20 | 4 | 0 | 0.0005 | 0.0000E+00 |
| HS38 | 4 | 49 | 179 | 6 | 0.0046 | 2.4984E-18 | 56 | 171 | 0.0049 | 3.7798E-13 |
| HS3MOD | 2 | 2 | 0 | 1 | 0.0004 | 7.8886E-31 | 4 | 4 | 0.0005 | 0.0000E+00 |
| HS4 | 2 | 1 | 0 | 0 | 0.0003 | 2.6667E+00 | 1 | 0 | 0.0003 | 2.6667E+00 |
| HS45 | 5 | 2 | 0 | 0 | 0.0004 | 1.0000E+00 | 2 | 0 | 0.0004 | 1.0000E+00 |
| HS5 | 2 | 10 | 12 | 1 | 0.0009 | -1.9132E+00 | 4 | 4 | 0.0005 | -1.9132E+00 |
| JNLBRNG1 | 9604 | 19 | 1746 | 0 | 15.4620 | -1.8057E-01 | 19 | 1750 | 15.5400 | -1.8057E-01 |
| JNLBRNG2 | 9604 | 11 | 1788 | 0 | 15.0310 | -4.1487E+00 | 11 | 1788 | 15.2270 | -4.1487E+00 |
| JNLBRNGA | 9604 | 18 | 1929 | 0 | 16.2910 | -2.7110E-01 | 18 | 1929 | 16.3300 | -2.7110E-01 |
| JNLBRNGB | 9604 | 7 | 5652 | 1 | 44.3230 | -6.3007E+00 | 7 | 4531 | 35.4890 | -6.3007E+00 |
| LINVERSE | 1999 | 11 | 1860 | 1 | 3.6007 | 6.8100E+02 | 23 | 1977 | 3.8677 | 6.8100E+02 |
| LOGROS | 2 | 42 | 36 | 7 | 0.0028 | 0.0000E+00 | 57 | 66 | 0.0036 | 0.0000E+00 |
| MAXLIKA | 8 | 8 | 37 | 3 | 0.0164 | 1.1493E+03 | 29 | 133 | 0.0593 | 1.1363E+03 |
| MCCORMCK | 5000 | 9 | 19 | 3 | 1.0651 | -4.5666E+03 | 5 | 11 | 0.3148 | -4.5666E+03 |
| MDHOLE | 2 | 54 | 85 | 5 | 0.0035 | 0.0000E+00 | 53 | 87 | 0.0035 | 0.0000E+00 |
| MINSURFO | 5002 | - | - | - | - | - | 13 | 929 | 5.3090 | 2.5069E+00 |
| NCVXBQP1 | 10000 | 4 | 0 | 0 | 84.3350 | -1.9855E+10 | 4 | 0 | 84.9720 | -1.9855E+10 |
| NCVXBQP2 | 10000 | 8 | 109 | 1 | 81.5080 | -1.3340E+10 | 8 | 109 | 80.9630 | -1.3340E+10 |
| NCVXBQP3 | 10000 | 9 | 186 | 1 | 57.0800 | -6.5594E+09 | 8 | 187 | 57.6960 | -6.5594E+09 |
| NOBNDTOR | 5184 | 23 | 1316 | 0 | 6.8315 | -4.4993E-01 | 23 | 1316 | 6.8255 | -4.4993E-01 |
| NONSCOMP | 5000 | 8 | 58 | 1 | 0.2486 | 1.8981E-14 | 8 | 44 | 0.2482 | 2.6382E-14 |
| OBSTCLAE | 9604 | 4 | 4734 | 0 | 42.5860 | 1.8865E+00 | 5 | 4795 | 43.1040 | 1.8865E+00 |
| OBSTCLAL | 9604 | 20 | 788 | 0 | 6.4845 | 1.8865E+00 | 20 | 788 | 6.5685 | 1.8865E+00 |
| OBSTCLBL | 9604 | 15 | 1651 | 0 | 21.0760 | 7.2722E+00 | 15 | 1686 | 21.5600 | 7.2722E+00 |
| OBSTCLBM | 9604 | 5 | 895 | 0 | 18.1620 | 7.2722E+00 | 5 | 895 | 18.1710 | 7.2722E+00 |
| OBSTCLBU | 9604 | 16 | 692 | 0 | 11.6760 | 7.2722E+00 | 16 | 692 | 11.7500 | 7.2722E+00 |
| ODNAMUR | 8 | 13 | 47821 | 10 | 317.6420 | 9.2366E+03 | 13 | 45604 | 308.6610 | 9.2366E+03 |

Table B.2: Numerical results for the *filter* and *pure trust-region* variants of FILTBOUND (continued on next page)

| | | *filter* variant | | | | | *pure trust-region* variant | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Problem | $n$ | # iter | # cgiter | # nfilt | CPU | $f(x^*)$ | # iter | # cgiter | CPU | $f(x^*)$ |
| OSLBQP | 11130 | 1 | 0 | 0 | 0.0004 | 6.2500E+00 | 1 | 0 | 0.0004 | 6.2500E+00 |
| PALMER1 | 4 | 24 | 15 | 3 | 0.0030 | 1.1755E+04 | 30 | 32 | 0.0038 | 1.1755E+04 |
| PALMER1A | 6 | 89 | 349 | 16 | 0.0150 | 8.9883E-02 | 188 | 801 | 0.0318 | 8.9883E-02 |
| PALMER1B | 4 | 33 | 59 | 16 | 0.0043 | 3.4473E+00 | 38 | 68 | 0.0048 | 3.4473E+00 |
| PALMER1E | 8 | 150 | 1357 | 10 | 0.0411 | 8.3523E-04 | 138 | 922 | 0.0315 | 8.3523E-04 |
| PALMER2 | 4 | 35 | 34 | 4 | 0.0039 | 3.6511E+03 | 17 | 27 | 0.0021 | 3.6511E+03 |
| PALMER2A | 6 | 45 | 147 | 9 | 0.0068 | 1.7110E-02 | 80 | 292 | 0.0114 | 1.7110E-02 |
| PALMER2B | 4 | 26 | 60 | 5 | 0.0031 | 6.2327E-01 | 23 | 59 | 0.0028 | 6.2327E-01 |
| PALMER2E | 8 | 183 | 1600 | 6 | 0.0417 | 2.0650E-04 | 231 | 1598 | 0.0450 | 2.0650E-04 |
| PALMER3 | 4 | 28 | 41 | 1 | 0.0034 | 2.4170E+03 | 26 | 32 | 0.0029 | 2.2660E+03 |
| PALMER3A | 6 | 64 | 228 | 15 | 0.0089 | 2.0431E-02 | 126 | 530 | 0.0181 | 2.0431E-02 |
| PALMER3B | 4 | 22 | 40 | 2 | 0.0025 | 4.2276E+00 | 26 | 67 | 0.0032 | 4.2276E+00 |
| PALMER3E | 8 | 110 | 969 | 10 | 0.0255 | 5.0741E-05 | 131 | 982 | 0.0264 | 5.0741E-05 |
| PALMER4 | 4 | 28 | 41 | 1 | 0.0034 | 2.4240E+03 | 25 | 35 | 0.0029 | 2.2854E+03 |
| PALMER4A | 6 | 42 | 152 | 7 | 0.0063 | 4.0606E-02 | 60 | 211 | 0.0085 | 4.0606E-02 |
| PALMER4B | 4 | 35 | 76 | 7 | 0.0041 | 6.8351E+00 | 26 | 64 | 0.0031 | 6.8351E+00 |
| PALMER4E | 8 | 133 | 1127 | 19 | 0.0305 | 1.4800E-04 | 113 | 807 | 0.0239 | 1.4800E-04 |
| PALMER5A | 8 | - | - | - | - | - | - | - | - | - |
| PALMER5B | 9 | 419 | 3992 | 30 | 0.0885 | 9.7525E-03 | - | - | - | - |
| PALMER5D | 8 | 2 | 7 | 2 | 0.0005 | 8.7339E+01 | 8 | 17 | 0.0010 | 8.7339E+01 |
| PALMER5E | 8 | 749 | 4607 | 50 | 0.1178 | 2.2113E-02 | - | - | - | - |
| PALMER6A | 6 | 227 | 990 | 16 | 0.0283 | 5.5949E-02 | 214 | 815 | 0.0265 | 5.5949E-02 |
| PALMER6E | 8 | 38 | 355 | 5 | 0.0083 | 2.2395E-04 | 118 | 833 | 0.0206 | 2.2395E-04 |
| PALMER7A | 6 | - | - | - | - | - | - | - | - | - |
| PALMER7E | 8 | 664 | 4245 | 16 | 0.1060 | 1.0154E+01 | 566 | 3019 | 0.0849 | 1.0154E+01 |
| PALMER8A | 6 | 49 | 184 | 16 | 0.0061 | 7.4010E-02 | 50 | 124 | 0.0059 | 7.4010E-02 |
| PALMER8E | 8 | 35 | 263 | 6 | 0.0065 | 6.3393E-03 | 60 | 385 | 0.0100 | 6.3393E-03 |

Table B.3: Numerical results for the *filter* and *pure trust-region* variants of FILTBOUND (continued on next page)

Table B.4: Numerical results for the *filter* and *pure trust-region* variants of FILTBOUND

| | | *filter* variant | | | | | *pure trust-region* variant | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Problem | $n$ | # iter | # cgiter | # nfilt | CPU | $f(x^*)$ | # iter | # cgiter | CPU | $f(x^*)$ |
| PENTDI | 5000 | 1 | 0 | 0 | 0.0173 | -7.5000E-01 | 1 | 0 | 0.0174 | -7.5000E-01 |
| PROBPENL | 500 | 1 | 0 | 0 | 0.0788 | 3.9920E-07 | 1 | 0 | 0.0788 | 3.9920E-07 |
| PSPDOC | 4 | 14 | 28 | 6 | 0.0013 | 2.4142E+00 | 6 | 16 | 0.0007 | 2.4142E+00 |
| QR3DLS | 610 | 137 | 134882 | 21 | 166.3570 | 1.1050E-10 | 212 | 18794 | 24.9150 | 1.0985E-10 |
| QRTQUAD | 5000 | - | - | - | - | - | - | - | - | - |
| QUDLIN | 5000 | 4 | 0 | 0 | 30.9030 | -1.2500E+09 | 4 | 0 | 25.3180 | -1.2500E+09 |
| S368 | 8 | 4 | 5 | 0 | 0.0010 | -6.2500E-01 | 4 | 5 | 0.0010 | -6.2500E-01 |
| SCON1DLS | 5000 | - | - | - | - | - | - | - | - | - |
| SIM2BQP | 1 | 1 | 0 | 0 | 0.0003 | 0.0000E+00 | 1 | 0 | 0.0003 | 0.0000E+00 |
| SIMBQP | 2 | 1 | 0 | 1 | 0.0003 | 0.0000E+00 | 4 | 0 | 0.0005 | 0.0000E+00 |
| SINEALI | 1000 | 6 | 39 | 0 | 0.0370 | -9.9901E+04 | 6 | 39 | 0.0370 | -9.9901E+04 |
| SPECAN | 9 | 10 | 74 | 2 | 0.5989 | 1.6600E-13 | 10 | 61 | 0.5451 | 1.6469E-13 |
| TORSION1 | 5184 | 23 | 815 | 0 | 3.9717 | -4.3028E-01 | 23 | 815 | 4.0007 | -4.3028E-01 |
| TORSION2 | 5184 | 9 | 808 | 0 | 6.5025 | -4.3028E-01 | 9 | 808 | 6.5215 | -4.3028E-01 |
| TORSION3 | 5184 | 11 | 224 | 0 | 1.0275 | -1.2170E+00 | 11 | 224 | 1.0157 | -1.2170E+00 |
| TORSION4 | 5184 | 9 | 347 | 0 | 8.5040 | -1.2170E+00 | 9 | 347 | 8.6290 | -1.2170E+00 |
| TORSION5 | 5184 | 6 | 75 | 0 | 0.3584 | -2.8634E+00 | 6 | 75 | 0.3672 | -2.8634E+00 |
| TORSION6 | 5184 | 5 | 96 | 0 | 6.7630 | -2.8634E+00 | 5 | 96 | 6.7945 | -2.8634E+00 |
| TORSIONA | 5184 | 23 | 817 | 0 | 4.2703 | -4.1830E-01 | 23 | 817 | 4.2707 | -4.1830E-01 |
| TORSIONB | 5184 | 9 | 875 | 0 | 7.1060 | -4.1830E-01 | 9 | 875 | 7.1420 | -4.1830E-01 |
| TORSIONC | 5184 | 11 | 224 | 0 | 1.0995 | -1.2042E+00 | 11 | 224 | 1.1121 | -1.2042E+00 |
| TORSIOND | 5184 | 9 | 332 | 0 | 9.0220 | -1.2042E+00 | 9 | 332 | 9.1570 | -1.2042E+00 |
| TORSIONE | 5184 | 6 | 75 | 0 | 0.3861 | -2.8502E+00 | 6 | 75 | 0.3969 | -2.8502E+00 |
| TORSIONF | 5184 | 5 | 96 | 0 | 7.2495 | -2.8502E+00 | 5 | 96 | 7.2455 | -2.8502E+00 |
| WEEDS | 3 | 34 | 59 | 4 | 0.0033 | 2.5873E+00 | 35 | 62 | 0.0038 | 2.5873E+00 |
| YFIT | 3 | 39 | 90 | 14 | 0.0043 | 4.8261E-10 | 68 | 172 | 0.0076 | 6.7413E-13 |